COMPLEXITY AND MODELING POWER OF INSERTION-DELETION SYSTEMS

Alexander Krassovitskiy

Dipòsit Legal: T-1370-2011

Universitat Rovira i Virgili

Departament de Filologies Romániques

Alexander Krassovitskiy

Complexity and Modeling Power of Insertion-Deletion Systems

PhD Dissertation

Supervised by:

Yurii Rogozhin

and

Sergey Verlan

Tarragona, 2011

Supervisors:

Professor Yurii Rogozhin

Senior researcher at the Institute of Mathematics and Computer Science

Academy of Sciences of Moldova

Str. Academiei, 5

MD-2028 Chişinău

MOLDOVA

Dr. hab. Sergey Verlan

Associated professor at Département Informatique

Université Paris Est

61, Av. Général de Gaulle

94010 Créteil

FRANCE

Tutor:

Dr. Gemma Bel-Enguix

Rovira i Virgili University

Research Group on Mathematical Linguistics

Av. Catalunya 35

43002 Tarragona

SPAIN

Abstract

The central notion of the thesis are insertion-deletion systems and their computational power. More specifically, we study language generating models that use two string rewriting operations: contextual insertion and contextual deletion.

We approach the questions about the minimal sizes of the insertion-deletion systems for normal and graph-controlled case. We show that the size of a system is one of the main parameters determining the computational power of classes of insertion-deletion systems. Our study consists of four parts, where we study the power of insertion only, insertion-deletion, graph-controlled, and graph-controlled systems with priorities, respectively.

In the first part we present equalities between context-free languages and the languages obtained by insertion systems with one-letter contexts and with specific squeezing mechanism. We prove the equivalence of these systems and the matrix languages if a graph-controlled variant is used. We also improve the result from [35] about the minimal size of computationally complete insertion systems by considering them in a graph-controlled framework. We also present some results concerning the semilinearity of Parikh sets of insertion systems.

In the second part we study one-sided and symmetrical insertion-deletion systems. We solve the last open problem regarding computational completeness of symmetrical insertion-deletion systems. We introduce and widely use the method of direct simulation for proving inclusions of families of languages generated by the insertion-deletion systems. We apply this method to a series of one-sided insertion-deletion systems and prove the equivalences of their computational power to Turing machines. We also found that some classes of insertion-deletion systems are not computationally complete. In the last two parts we study such systems in a graph-controlled framework, and we show that by this technique the generative power is strictly increased. At the end, we discuss some open problems raised by our investigations.

Acknowledgments

It is my pleasure to thank the many people who helped me during this thesis.

First of all, I would like to express sincere gratitude to my supervisor Professor Yurii Rogozhin who made this work possible. I am in debt to him for providing me with so many ideas and giving me a lot of inspiration and motivation.

Then, I would like to express my deep gratitude to my second supervisor Dr. hab. Sergey Verlan. Research visits organized by Sergey have been always very fruitful. I would also like to thank him for his expertise and patience, not to mention the proofreading of this work.

I would like to thank the head of our research group, Professor Carlos Martín-Vide, for his great work in establishing the wonderful environment of the International PhD School in Grammars, Formal Languages and Applications, and for his help in receiving financial support throughout my PhD. This thesis was made thanks to the financial support of research grant Ramon i Cajal from the University Rovira i Virgili, and in part by the project MTM2007-63422 from the Spanish Ministry of Science and Technology.

I would like to address especial thanks to Dr. Gemma Bel Enguix, who kindly accepted to be the tutor of my thesis, and for her worthy comments on it. I appreciate a lot that she organized and provided funding for me, necessary to attend different conferences.

I also address my warm thanks to all professors of the PhD school in Formal Languages and Applications for presenting so many wonderful topics in Theoretical Computer Science, and who contributed so much to the organization of the school.

I would like to extend my gratitude to my colleagues Sherzod Turaev, Aretí

Panou, Adrian Horia Dediu, Madalina Barbaiani, Cătălin Tîrnăucă, Cristina Tîrnăucă, Guangwu Liu, Alexander Perekrestenko, Artiom Alhazov, Peter Leupold and to all the students of Research Group on Mathematical Linguistics of University Rovira i Virgili with whom I could feel like at home.

I would like to thank Dr. Lilica Voicu, who has been always very helpful and very rapid in solving so many administrative problems during all my stay in Tarragona.

I also would like to thank Professor Semen Serovajskiy, whose lectures and discussions gave me the chance to admire the mathematics and follow the research in Computer Science.

Last but not the least I wish to express sincerely thanks my parents for their deep understanding and patience.

Contents

List of Figures

Introduction

What could be the most generic tools that helps humans to go through daily obstacles? A possible answer could be described in terms of languages, and information they pass. We could claim that in every scientific discipline, the understanding of a subject can be done only by understanding its proper language. Formal languages often work as a tool that allow researcher to understand in an unambiguous way the properties of desired system. For example, in order to create a software or hardware one needs a lot of investigations into the subject areas. This, in turn, requires to describe formally particular components to be implemented, and, of course, the selection of an adequate language.

Up to now the theory of formal languages has provided formalisms for plenty of disciplines of human society. We elaborate our thesis around two most interesting operations of formal languages called *insertion and deletion*.

The operations of insertion and deletion have a long history. Many linguists have used various types of string insertion in order to model properties of natural languages. For example, in Marcus contextual grammars string contextual insertion are used [47, 24, 58].

Another inspiration for insertion-deletion operation comes from formal language theory. The insertion operation and its iterated variants are generalized versions of Kleene's operation of concatenation [38], while the deletion operation generalizes the quotient operation. A study of properties of the corresponding operations may be found in [26, 27, 31].

The third motivation for the study of insertion and deletion comes from the field of molecular biology. The experimental biology collects a large amount of information about living matter: genes, biological functions, cells, etc. It is clear that the variety of the biological data needs systematization and understanding. Computer science have given plenty of efforts in order to understand the functionality of the living matter. This eventually helps to promote the means of regulations and controls into medicine, biology, and human society. In order to treat the great amount of raw data formal models are needed. Recently, it was shown that insertion and deletion correspond to a mismatched annealing of DNA sequences. Such operations are also present in the evolution processes in the form of point mutations as well as in RNA editing, see the discussions in [12, 13, 64] and [62]. This biological motivation of insertion-deletion operations led to their study in the framework of molecular computing, see, for example, [18, 33, 62, 65].

In general, an insertion operation means adding a substring to a given string, while a deletion operation means removing a substring of a given string. In our research we mainly consider *contextual string insertion* and *contextual string deletion* operations, *i.e.*, inserted and deleted in specified (left and right) contexts. A contextual insertion or contextual deletion rule is defined by a triple (u, x, v) meaning that x can be inserted between u and v or deleted if it is between u and v. Thus, an insertion corresponds to the rewriting rule $uv \to uxv$ and a deletion corresponds to the rewriting rule $uxv \to uv$. Further we omit term *contextual*, whenever it is clear from the presentation. A finite set of insertion-deletion rules, together with a set of axioms provide a language generating device: starting from the set of initial strings and iterating insertion or deletion operations as defined by the given rules one gets a language. The size of the alphabet, the number of axioms, the size of contexts and of the inserted or deleted string are natural descriptional complexity measures for insertion-deletion systems.

The size of an insertion-deletion system is represented as a 6-tuple (n, m, m', p, q, q'), where n is a maximal length of the inserted string, m, m' are the maximal lengths of the left and right contexts over all insertion rules, p is a maximal length of the deleted string, and q, q' are the maximal lengths of the left and right contexts over all deletion rules. Our research studies those families of languages which are generated by insertion-deletion systems with different sizes, in particular, those insertion-deletion systems having some of m, m', q, q' equal to zero. It appears that the notion of size becomes one of the central points of the thesis as it is the main parameter of descriptional complexity of the system. We investigate relation between the computational power of insertion-deletion systems and their size. Such approach allows to find the borderline between computationally complete and computationally uncomplete systems. We solved many challenging questions related to the minimal sizes of the insertion and deletion systems for both graph-controlled and pure systems. While several interchanges between these parameters allowed to estimate the borderline for the computationally completeness of our model, for the uncomplete systems the positions of the generated families in the Chomsky hierarchy are studied.

The thesis is organized as follows. Chapter 1 gives a brief introduction into the history of insertion and deletion systems. Here we introduce the main research areas which gave the inspiration for writing the thesis. The most significant previous results are summarized in this chapter.

In order to make the thesis accomplished and to fix the notations we introduce in Chapter 2 the general definitions from formal language theory which are used throughout the thesis.

Chapter 3 is devoted to pure insertion systems and graph-controlled insertion systems (all these systems do not use the deletion operation). We present several new equivalences between the language families in Chomsky hierarchy and the insertion systems. All these results improve known results from the literature. The material of this chapter is based on [8, 39, 40].

Chapter 4 deals with systems where both operations of insertion and deletion are used. Firstly we consider systems where sizes of left and right contexts are equal and we give an important result concerning the computational completeness of insertion-deletion systems of size $(1, 1, 1; 2, 0, 0)$.

In this chapter we also investigate *one-sided* insertion-deletion systems, *i.e.* systems whose rules have a left (or right) context only. We show that a one-sided system can be Turing equivalent, if the sizes of contexts of its rules are sufficiently large. In this chapter we also give the idea of methods used in the following proofs.

We introduce a new method of computational completeness proofs for insertion-deletion systems that permitted to prove most of the results from this chapter. We also show a series of computational uncompleteness and decidability results. The results from this chapter are based on [42, 43, 44]. These results are summarized in Table 4.1 and Table 4.2.

In Chapter 5 we consider graph-controlled insertion-deletion systems. We take insertion-deletion systems which are shown to be uncomplete and consider them in a graph-controller manner. We show that the computational power strictly increases in these cases. We note, that these systems have their origins from insertion-deletion P systems firstly considered in [62]. The material of this chapter is based on [7, 41, 43, 44].

The final chapter is devoted to graph-controlled insertion-deletion systems with priorities corresponding to graph-controlled systems with appearance checking. More precisely, having an insertion and a deletion rules applicable, then always the deletion is chosen to be applied. The results on the computational complexity are given for the classes of languages which can be generated by systems of very small sizes. We show that this type of priorities is extremely powerful. The results from this chapter are mostly based on [6, 8, 9].

Chapter 1

State of the art

This chapter gives a short introduction into the history of insertion-deletion systems. Firstly, we will present the linguistic origins of insertion and deletion operations. Then, a formal languages motivation is presented. Finally, we give the biological background for insertion-deletion systems.

1.1 Linguistic motivation

The idea of insertion of one string into another was firstly considered with a linguistic motivation by Marcus in [47]. It is known that a Chomsky grammar is not a unique way to represent natural languages. Moreover, for natural languages the models that use local insertion/deletion may be simpler than models that use top-down grammatical tree construction of words for a given language. For example, in an English sentence a noun phrase permits the insertion of an arbitrary number of adjectives adjacently. This can be represented in a simplified form by an insertion rule (a, a', n), where $a, a' \in ADJECTIVES, n \in NOUNS$. Clearly, the above rule is easier to express in terms of locality than by syntactical tree.

We would like to cite Marcus concerning the idea that stays behind contextual string insertion: *"Generative grammars are a rupture from the linguistic tradition of the first half of XX-th century, while analytical models are just the development, the continuation of this tradition... Contextual grammars have their origin in the attempt to transform some procedures developed within the framework of analytical* *model into generative devices. The idea of connecting in this way analytical study with the generative approach to natural languages was one of main problems investigated in mathematical linguistics in the period from 1957 to 1970."*

Marcus contextual grammars consider couples $(x,(u,v))$, meaning that words u and v can be adjoined to the word x. This corresponds in some sense to grammars having rules of type $x \rightarrow uxv$, *i.e.*, u and v are inserted around the position marked by x. Such grammars are considered as a generative device that permits insertions of the given contexts, in cases x satisfies certain conditions, e.g., written by a regular expression. Many interesting linguistic issues like ambiguity and word duplication can be captured in this framework.

This idea was later developed in [58, 48], and with particular application in [11]. The fixed size contexts (specified for every rule) were firstly considered in [24].

1.2 Formal languages motivation

In [26, 27] the insertion operation and its iterated variant are introduced with rather different motivation. The author considers these operations as generalization of Kleene's operations of concatenation and closure [38]. The operation of concatenation would produce a string x_1x_2y from two strings x_1x_2 and y. By allowing the concatenation to happen anywhere in the string and not only at its right extremity a string x_1yx_2 can be produced, *i.e.*, y is inserted into x_1x_2. In [31] the deletion is defined as a right quotient operation which happens not necessarily at the rightmost end of the string. In the same thesis the duality between the insertion and deletion is also highlighted: any insertion system generating a language $\mathcal{L}$ is at the same time a deletion system recognizing $\mathcal{L}$. The operations considered in above works correspond to context-free variants of insertion and deletion operations, because no contexts are used. In the same place several other variants of insertion and deletion are introduced and their closure properties are investigated.

In the literature it is possible to find several investigations on the extension of regular expressions with the operation of insertion. Formally, the extended regular expressions $ExReg$ are defined as the smallest family of languages which contains the finite languages and is closed under the operations of union, concatenation, Kleene star and iterated insertion. The article [27] shows that the power of $ExReg$ languages is strictly between regular and context-free languages. One of the main goals of the article is to find connections between these languages and context-free languages. In particular, it verifies whether well-known decidability results for context-free and regular languages hold as well for the insertion languages. For example, it studies the decidability problems of emptiness and regularity for an $ExReg$ language given by its rules as well as the context-freeness of the intersection of two $ExReg$ languages.

The article [29] studies the languages that are closed under insertion. It seems that the computational power of such languages is rather weak. This is partially due to the restriction that for every word $w \in ins(L)$, $ins(L)$ being the insertion closure of L, word w must be "insertable" into every word from L and moreover at every possible position, resulting again a word from L. Hence, L is closed with respect to insertion if it has some trivial form, for example, a^+, the Dyck language, etc.

One of the ways to study the operation of insertion is to apply it on the family of well-known languages from Chomsky hierarchy. In [31] several types of insertions are defined:

1. Sequential language insertion SIN of L_2 into L_1 over alphabet Σ, defined as: $SIN(L_1, L_2) = \{u_1vu_2 \mid u_1u_2 \in L_1, v \in L_2\}$.

2. Parallel language insertion PIN over of L_2 into L_1 over Σ, defined as: $PIN(L_1, L_2) = \{w \in \Sigma \mid w = a_1v_1a_2v_2 \dots v_{k_1}a_k, a_1, \dots, a_k \in \Sigma, a_1a_2 \dots a_k \in L_1, v_1, \dots, v_{k-1} \in L_2\}$.

3. Permuted sequential language insertion $PSIN$ of L_2 into L_1 over Σ, defined as: $PSIN(L_1, L_2) = \{u_1vu_2 \mid u_1u_2 \in L_1$, there is $v' \in L_2$ such that $v \in perm(v')\}$, where $perm(v)$ is set of permutations of v.

4. Permuted parallel language insertion $PPIN$ of L_2 into L_1 over Σ, defined as: $PPIN(L_1, L_2) = \{w \in \Sigma \mid w = a_1v_1a_2v_2 \dots v_{k_1}a_k, a_1, \dots, a_k \in \Sigma$, $a_1a_2 \dots a_k \in L_1, v_1, \dots, v_{k-1} \in \Sigma^*$, there are $v'_1, \dots, v'_{k-1} \in L_2$ such that $v_i \in perm(v'_i), 1 \leq i \leq k-1\}$.

5. Controlled sequential language insertion $CSIN$, defined via a control function $\Delta : \Sigma \to 2^{\Sigma^*}$. $CSIN(L_1, \Delta) = \{u_1avu_2 \mid u_1u_2 \in L_1, a \in \Sigma, v \in \Delta(a)\}$.

Table 1.1: Closures of language under insertions

Op	Reg	CF	CS	Op	Reg	CF	CS
SIN	YES	YES	YES	$PPIN$	NO	NO	YES
SIN_*	NO	YES	YES	$CSIN$	YES	YES	YES
PIN	YES	YES	YES	$CSIN_*$	NO	YES	YES
PIN_*	NO	NO	YES	$CPIN$	YES	YES	YES
$PSIN$	NO	NO	YES	$CPIN_*$	NO	NO	YES

6. Controlled parallel language insertion $CPIN(L_1, \Delta) = \{a_1v_1 \dots a_kv_k \mid a_1 \dots a_k \in L_1, v_i \in \Delta(a_i), i = 1, \dots, k\}$.

The reflexive and transitive closure of $Op \in \{SIN, PIN, CSIN, CPIN\}$ is denoted by $Op_*(L_1, L_2)$.

$$Op_0(L_1, L_2) = L_1,$$
$$Op_k(L_1, L_2) = Op(Op_{k-1}(L_1, L_2), L_2),$$
$$Op_*(L_1, L_2) = \bigcup_{n=0}^{\infty} Op_n(L_1, L_2).$$

The results for insertions of types $SIN, SIN_*, PIN, PIN_*, PSIN, PPIN, CSIN, CSIN_*, CPIN, CPIN_*$ are summarized in Table 1.1. Analogous results were obtained for deletion operations.

Further results on iterated (sequential) deletion were obtained in [20]. The article nicely presents a technique for encoding recursively enumerable languages by means of iterated (sequential) deletion and intersection with regular languages. Moreover, the intersection with the regular language is only needed to separate those words that begin and end with special markers. It also shows that the iterated deletion of linear context-free languages has certain limits in the generative power. In particular, there are simple regular languages that cannot be generated by any recursive deletion languages (when no additional intersection or coding is used). The main result of the article proves the following relation: for every regular language L there is linear context-free language L' and a regular language R such that $L = del(L') \cap R$, where $del(L')$ denotes the iterated closure of L' with respect to deletion.

Several investigations pointed out that insertion and deletion may be seen as particular case of string rewriting manipulations. Shuffle and deletion on trajectories was studied intensively in [34]. This model has a particular interest because it is easy to consider the operation of shuffle on trajectories as a generalization of insertion (with no contexts). Hence, many results that are applicable to systems based on operations of shuffle and deletion on trajectories are also applicable for the insertion and deletion systems. For example, as an immediate result one can obtain the closure of regular languages under the iterated insertion.

In the above article a list of decidability results is given for shuffle and deletion on trajectories for context-free and regular languages. In particular, it was shown that for any two regular languages it is decidable whether the language which is resulted by insertion or deletion of one language into another one is regular. Moreover, this language may be effectively constructed. In case of regular language being inserted into (or deleted from) a context-free language this problem is undecidable.

It was shown in [19] how the semantic shuffle and deletion along trajectories may be used as a generalized model for the contextual insertion and the contextual deletion. This model has a very clear form for an algorithmic implementation and can be used to describe many other binary string operations. In a unified form the techniques presented in the article help to solve many language equations. However, this technique cannot be applied for the iterative form of the computations (as in the case of derivations of insertion and deletion systems), because the generating power of such a model increases significantly, giving a Turing equivalent model. Further results for restricted variants of the shuffled insertion were obtained in [30].

It was proved in [62] that pure insertion systems having one letter context are always context-free. Yet, there are insertion systems with two letter context which generate nonsemilinear languages (see Theorem 6.5 in [62]). On the other hand, it appears that by using only insertion operations the obtained language classes with contexts greater than one are incomparable with many known language classes. For example, there is a simple linear language $\{a^nba^n \mid n \geq 1\}$ which cannot be generated by any insertion system (see Theorem 6.6 in [62]).

In order to overcome this obstacle one can use some codings to “interpret” the generated strings. The questions about the computational power of insertion systems with morphisms and intersection with special languages were considered in [55, 56] and [61]. In [50] two additional mapping relations are used : a morphism h and a weak coding φ. The strings of the language are obtained by applying $h^{-1} \circ \varphi$ on the generated strings. Clearly, the languages obtained in such a way have greater expressivity, and the corresponding language class is more powerful. It appears that in this case one can obtain every RE language if insertion rules have sufficiently large context.

We define the size of an insertion system as a vector $(n, m, m'), n > 0, m, m' \geq 0$, where n is a maximal length of the inserted strings; m and m' are equal to the maximal length of the left and the right contexts of rules of the system. This vector corresponds to the first three parameters in the definition of size for insertion-deletion systems. It is proved in [50] that for every recursively enumerable language L there exists a morphism h, a weak coding φ and a language L' generated by an insertion system with rules having sizes at most $(7, 7, 7)$, such that, $L = h(\varphi^{-1}(L'))$. This result was improved in [54], where it was shown that systems having rules of size at most $(5, 5, 5)$ are sufficient to encode every recursively enumerable language. This result was further improved in [70]. Recently, in [35] it was shown that the same result can be obtained with rules of size equal to $(3, 3, 3)$. We improve this result by introducing a graph control into the model. In this case the computational completeness with rules of size $(2, 2, 2)$ is obtained, see Theorem 3.3.7.

Article [36] introduces the operations of contextual insertion and contextual deletion as generalizations of insertion and deletion of words. Closure properties of the regular and context-free languages under these operations, contextual ins-closed and del-closed languages, and decidability of existence of solutions to equations involving these operations are investigated. Based entirely on contextual operations, the insertion and deletion systems have been introduced, where both types of rules can act simultaneously in the same derivation. Moreover, in this article it was shown that every Turing machine can be simulated by insertion-deletion systems. This work also introduces the notion of lengths of contexts as basic computational parameter, called weight. It corresponds to the 4-tuple $(n, \bar{m}, p, \bar{q})$, where $\bar{m} = max\{m, m'\}$, and $\bar{q} = max\{q, q'\}$, where (n, m, m', p, q, q') is the size of the system. Since this work a number of studies have been done in this direction. For example, an attempt to use the number of symbols of the alphabet as a measure of the (descriptional) complexity was given in [37]. It is shown there that two symbols are enough to obtain the power of Turing machine.

We would like to remark one result from [49] where it was proved that even relatively small sized insertion-deletion systems which do not use contexts are computationally complete. In fact, this article shows that for any type-0 grammar there exists an insertion-deletion system of size $(n, 0, 0; m, 0, 0)$ which generates the same language, where parameters n and m depend on the form of the used grammar. Then, the result was improved for fixed size insertion-deletion systems by using special normal forms of RE grammars. More precisely, the inclusions $RE \subseteq INS_3^{0,0}DEL_2^{0,0}$ and $RE \subseteq INS_2^{0,0}DEL_3^{0,0}$ were shown. The article [67] shows that similar results do not hold for a system of smaller size.

Table 1.2 contains the best known results on complexity of insertion-deletion systems and Table 1.3 contains results for non-symmetrical insertion-deletion systems.

Table 1.2: Known results on insertion-deletion systems

Nb.	$(n, m, m'; p, q, q')$	size	family	references
1	$(2, 0, 0; 3, 0, 0)$	5	RE	[49]
2	$(3, 0, 0; 2, 0, 0)$	5	RE	[49]
3	$(1, 1, 1; 2, 0, 0)$	5	RE	[62], Theorem 4.3.2
4	$(1, 1, 1; 1, 1, 1)$	6	RE	[65, 66]
5	$(2, 0, 0; 2, 0, 0)$	4	$\subsetneq CF$	[67]
6	$(m, 0, 0; 1, 0, 0)$	m+1	$\subsetneq CF$	[67]
7	$(1, 0, 0; p, 0, 0)$	p+1	$\subsetneq REG$	[67]

It is clear from Table 1.2 that the generating power of symmetrical insertion-deletion systems has been already described for almost all combinations of parameters. Because of this we continued the investigation of one-sided insertion-deletion systems. The complete list of obtained results can be found in Tables 4.1 and 4.2.

Table 1.3: Known results on one-sided insertion-deletion systems

1	$(1,1,2;1,1,0)$	6	RE	[51]
2	$(2,0,2;1,1,0)$	6	RE	[51]
3	$(2,0,1;2,0,0)$	5	RE	[51]
4	$(1,1,1;1,1,0)$	5	$\subsetneq RE$	[51]

The insertion-deletion operations were considered in the framework of molecular computing [60, 16, 57, 59]. The definition of insertion and deletion system presented there makes a general assumption that there are disjoint groups of insertion and deletion rules (corresponding to *membranes* from [60] or *components* from [16]) which work in a specific order defined by a graph control.

Some attempts to classify insertion-deletion P systems in terms of Chomsky hierarchy may be found in [46, 45, 23]. In the presented works the families of insertion-deletion P systems have been defined according to the maximal sizes of insertion and deletion rules. In addition, there are two complexity parameters that specify the depth and the size of membrane tree structure. The presented results compare the computational power of insertion-deletion P systems with the families of context-free, matrix and recursively-enumerable languages.

In this thesis we have worked on the simplification of the definition of insertion-deletion P systems and presented there as a particular case of a graph-controlled scheme.

1.3 Biological motivation

The third inspiration for insertion and deletion operation comes, surprisingly, from the field of molecular biology. We present the general idea about how insertion and deletion systems can be seen as a formalization of DNA and RNA processes corresponding to a mismatched annealing of DNA sequences. Investigations in *DNA computing* had received significant inspiration after the publication of Adleman [1] in 1994, where it was shown that some important mathematical problems can be solved by means of DNA molecules. The experiment conducted by Adleman have used DNA molecules in order to solve an instance of well-known NP-complete problem called *Hamiltonian path* problem. Since then a big amount of theoretical and practical investigations has been done giving a growth for such areas as insertion-deletion (P) systems, splicing systems, sticker systems, Head systems, Watson-Crick automata, etc., see [62]. We present below how it is theoretically possible to perform the insertion and the deletion involving molecules of DNA and RNA.

1.3.1 Motivation from DNA computing

We give the description of (contextual) insertion-deletion systems in terms of operations on DNA sequences taken from [62]. Traditionally a DNA sequence is represented as an oriented string with its leftmost end marked by $5'$ and its rightmost end marked by $3'$. The alphabet consists of four letters $\{A, T, G, C\}$ for adenine, thymine, guanine, and cytosine. Complementary symbols are denoted by bared symbols: $\overline{A}, \overline{T}, \overline{G}, \overline{C}$, and $\overline{A} = T, \overline{T} = A, \overline{G} = C, \overline{C} = G$. For further biological terminology see [62, 2]. Insertion and deletion can be performed, at least theoretically, as follows. Let us imagine that there is a test tube with a single stranded DNA sequence $5' - w_1uvw_2z - 3'$. If one adds to the test tube a single stranded DNA sequence $3' - \overline{u\alpha v} - 5'$, where $\overline{u}$,$\overline{v}$ are the Watson-Crick complements of strings u,v then the two strings might anneal (u will stick to $\overline{u}$ and v will stick to $\overline{v}$, folding α, see Fig. 1.1(b). Now one can cut the sequence uv obtaining the structure depicted in Fig. 1.1(c). Adding a primer $\overline{z}$ and the polymerase the complete double-stranded sequence is obtained, see Fig. 1.1(d). Finally, melting the solution the strands are separated leading to situation depicted in Fig. 1.1(e), meaning that α was inserted between u and v.

By a similar mismatched annealing one can, theoretically, perform a deletion operation, taking $u\alpha v$ in the starting string and adding $\overline{uv}$. The process is illustrated in Fig. 1.2 (in order to go from step (b) to step (c) a polymerization and removing of the loop by a restriction enzyme is done).

In addition to the fact that insertion and deletion can be performed in the DNA framework such operations are also present in the evolution processes under the form of point mutations, see the discussions in, [62] and [64]. Presented biological

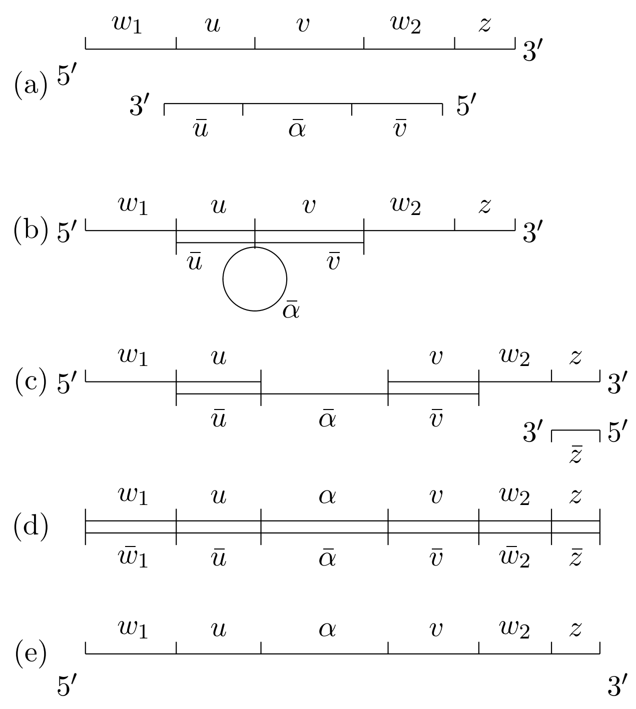


Figure 1.1: Inserting by mismatched annealing

motivation of insertion-deletion systems lead to their study in the framework of molecular computing as, for example, in [18, 32, 33, 62, 69], and [65].

Another model that uses insertion and deletion operations is inspired by cooperative strategies observed in micro-biology. This approach investigates elementary processors and the cooperation among them in a network (of evolutionary processors). Processors of a network can perform elementary string operations: insertions, deletions, and substitutions. The model was introduced in [15] and further studied in, e.g., [10, 53]. Some variations of evolutionary network systems were presented in [3, 17] (for hybrid systems), and in [4] (for obligatory hybrid systems).

1.3.2 Motivation from RNA editing

Guided insertion-deletion systems have been considered in [14, 13]. The main object of investigations in this article is a protozoa called *Kinetoplastid*. Some biological phenomena that occur during RNA editing have given a motivation to consider

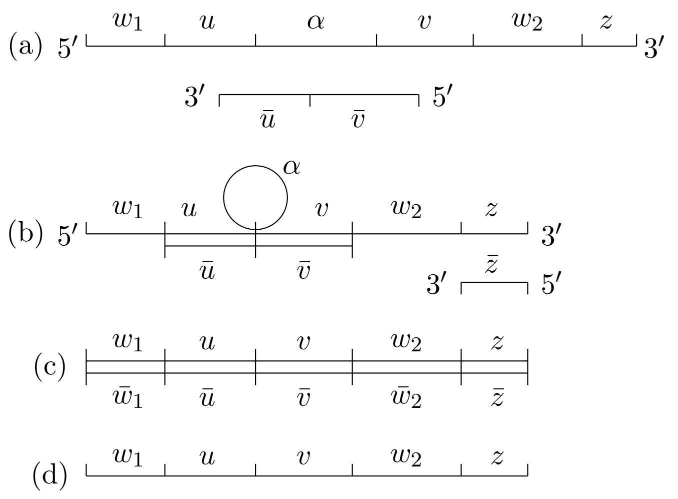


Figure 1.2: Deleting by mismatched annealing

an extension of the contextual insertion and deletion systems. In this model the matrix and the guided RNA are represented by two separate strings. Every step that modify the string for matrix RNA has to be done in a correspondence with some modifications in the string for the guided RNA. For example, let (a, ccc) be a pair of strings that corresponds to the matrix and to the guided RNAs; let (a, b, ε) and (c, d, c) be a pair of insertion rules that are present in the matrix and the guided RNA, correspondingly. Then, we have following derivation: $(a, ccc) \Rightarrow (ab, cdcc) \Rightarrow (abb, cdcdc)$.

The system presented in [13] can be considered as a merge of two insertion and deletion systems. When one system is evolved the other system is also necessarily evolved. Clearly, this model of computation generalizes the insertion and deletion model: two models are equal if the guided rules corresponding to the second system are trivial and do not impose any restrictions on the application of the rules, e.g., all the guided rules have form $(\varepsilon, \varepsilon, \varepsilon)$. We note that depending on the form of guided rules one may distinguish a *semi-constant* insertion and deletion system (the corresponding family of languages is denoted by $SCGI$) if for every insertion and deletion rule $z = \varepsilon$. Also, it is possible to distinguish *uniting* ($G_1 = G \cup \{y_1\}$) and *non-uniting* ($G_1 = G \cup \{y_1\} \backslash \{y\}$) modes of the system. The language families corresponding to such guided insertion and deletion systems are denoted by uGI and $unGI$.

It is known from [13] that the generative capacity of such systems forms a Chomsky-like hierarchy of languages

$$Reg \subset SCGI \subset uGI \subset unGI \subset RE.$$

Moreover, the families of languages $SCGI$ and uGI are anti-AFL. From the other side [14] demonstrates how RNA editing may be accurately modeled by the guided insertion and deletion systems.

Another form of guided operations of insertion and deletion inspired by RNA editing was considered in [71]. The author observes that uracil (denoted by 0) is the only element which is inserted or deleted during RNA transcriptions. The transition step $\Rightarrow_G$ of the corresponding system is defined by using a set of guides $G \subset V^*$, where V is the working alphabet. We have $usv \Rightarrow_G ugv, u, v, s \in V^*, g \in G$ if g is obtained from s either by insertion or by deletion of one ore more occurrences of $0 \in V$. For example, given a sentential form $a00a0a0a$ and a set of guides $G' = \{a0a00a, aaa\}$. Then we have $a00a0a0a \Rightarrow_{G'} w, w \in \{a00a0a00a, a0a00aa, a00aaa, aaa0a\}$. The main result of this article states that the regular and the context-free languages are not closed under the operations of guided insertion and deletion.

More details about RNA editing for various biological species may be found, e.g., in [12].

Chapter 2

Preliminaries

This chapter introduces the general definitions from the theory of formal languages, used later in the thesis. Many definitions in this chapter are standard and may be found in any textbook on formal languages (e.g. [63, 28, 21, 25]).

2.1 Grammars, automata, and formal 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 2^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* or *symbols*. A *word* over the alphabet V is a concatenation of symbols of V. Sometimes we use term *string* instead of word. 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.

We denote by $|w|$ the length of a word w. For a letter a and a word w we denote by $|w|_a$ the number of letters a in w. We extend this notation to $|w|_V$, where V is an alphabet, which gives the number of letters from V in w. If A is a set of words, then we put $|A| = \max_{w \in A} |w|$. By $alph(w)$ we denote the set of letters occurring in w.

For a word $w \in V^*$ we define $Perm(w) = \{w' \mid |w'|_a = |w|_a \text{ for all } a \in$

$V\}$, and we denote by $\sqcup\!\sqcup$ the binary shuffle operation. We recall that $x \sqcup\!\sqcup y = \{x_1y_1 \dots x_ny_n \mid x = x_1 \dots x_n, y = y_1 \dots y_n, x_i, y_i \in V^*, 1 \le i \le n\}$.

In the sequel we will use some normal forms for context-free and type-0 grammars.

Definition 2.1.1. A context-free grammar $G = (N, T, S, P)$ is said to be in *Chomsky normal form* if it has rules of form $A \to BC$, $A \to x$, where $A, B, C \in N$, $x \in T$.

Definition 2.1.2. A type-0 grammar $G = (N, T, S, P)$ is said to be in *Pentonnen normal form* if it has rules of form $AB \to AC$, $A \to x$, where $A, B, C \in N$, A, B, C being different, $x \in (N \cup T)^*$ and $|x| \le 2$. We can also assume that x is either ε or equal to uv, where $u, v \in N \cup T$ and $A \ne u$, $u \ne v$, $A \ne v$.

Definition 2.1.3. A type-0 grammar $G = (N, T, S, P)$ is said to be in *special Pentonnen normal form* if it has rules of form $AB \to AC$, $BA \to CA$, $A \to AB$, $A \to BA$, $A \to \delta$, where $A, B, C \in N$ are different and $\delta \in N \cup T \cup \{\varepsilon\}$.

Definition 2.1.4. A type-0 grammar $G = (N, T, S, P)$ is said to be in *Kuroda normal form* if it has rules of form $AB \to CD$, $A \to BC$, $A \to \delta$, where $A, B, C, D \in N$ and $\delta \in T \cup \{\varepsilon\}$.

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

$$G = (\{S\}, T_n, S, \{S \to \varepsilon, S \to SS\} \cup \{S \to a_i S \bar{a}_i \mid 1 \le i \le n\}).$$

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

We also recall the following definition from [60]. A *context-free matrix grammar* (without appearance checking) is a construct $G = (N, T, S, M)$, where N, T are disjoint alphabets (of non-terminals and terminals, respectively), $S \in N$ (axiom), and M is a finite set of matrices, that is sequences of the form $(A_1 \to x_1, \dots, A_n \to x_n), n \ge 1$, of context-free rules over $N \cup T$. For a string w, a matrix $m : (r_1, \dots, r_n)$

is executed by applying the productions $r_1, \dots, r_n$ one after the other, following the order in which they appear in the matrix. Formally, we write $w \Rightarrow_m u$ if there is a matrix $m : (A_1 \to u_1, \dots, A_n \to u_n) \in M$ and strings $w_1, w_2, \dots, w_{n+1} \in (N \cup T)^*$ such that $w = w_1, w_{n+1} = u$, and for each $i = 1, 2, \dots, n$ we have $w_i = w' A_i w''$ and $w_{i+1} = w' u_i w''$. If the matrix m is understood, then we write $\Rightarrow$ instead of $\Rightarrow_m$. As usual, the reflexive and transitive closure of this relation is denoted by $\Rightarrow^*$. Then, the language generated by G is $L(G) = \{w \in T^* \mid S \Rightarrow^* w\}$. The family of languages generated by context-free matrix grammars is denoted by MAT^λ. It is well-known fact that every language $L \in MAT^\lambda$ can be generated by a matrix grammar in binary normal form $G' = (N \cup Q \cup \{S'\}, T, S', M')$, such that $L = L(G')$, where $Q = \{q_0, ..., q_m\}, m \geq 0$, $N \cap Q = \emptyset$, matrices $M' = M \cup \{m_0 : (S' \to Sq_0)\}$ having each matrix $m \in M$ of the following form $m : (q \to q', A \to \alpha)$, for $q, q', A' \in N, \alpha, \in (N \cup T)^*, |\alpha| \leq 2$. Sometimes it is handy to use a modified binary normal form similarly to the binary normal form, see e.g., [60], having each matrix m of the following form $m : (A \to \alpha, A' \to \alpha')$, for $A, A' \in N, \alpha, \alpha' \in (N \cup T)^*, max(|\alpha|, |\alpha'|) \leq 2$.

The family of recursively enumerable languages is denoted by RE. The Parikh image of a language family F is a family of sets of vectors denoted by PsF (we assume a fixed ordering on the alphabet $T = \{a_1, \dots, a_n\}$):

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

Definition 2.1.5. A set $S \in N^k$ is *linear* if S can be represented in the form

$$S = \{(x_1, \dots, x_k) + \sum_{l=1,\dots,s} k_l \cdot (x_{l_1}, \dots, x_{l_k}) \mid k_l \geq 0\}.$$

Definition 2.1.6. A set $S \in N^k$ is *semilinear* if S is a finite union of linear sets.

Definition 2.1.7. A *finite automaton* (see, e.g., [21]) is the quintuple $\mathcal{A} = (Q, V, q_0, F, \delta)$, 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
- $\delta : Q \times V \to 2^Q$ is the transition function of the automaton.

We define the transition $\to$ in an ordinary way: $(q, aw) \to (q', w)$ if $q' \in \delta(q, a)$, where $q, q' \in Q$, $a \in V$ and $w \in V^*$. We denote by $\to^*$ the reflexive and transitive closure of $\to$.

We say that the word w is accepted by $\mathcal{A}$ if $(q_0, w) \to^* (q, \varepsilon)$ and $q \in F$. The language accepted by $\mathcal{A}$ is:

$$L(\mathcal{A}) = \{w \in V^* \mid (q_0, w) \to^* (q, \varepsilon), q \in F\}.$$

Definition 2.1.8. A *register machine* (introduced in [52], see also [22]) is a construct

$$\mathcal{M} = (d, Q, q_0, h, P)\,,$$

where

- d is the number of registers,
- Q is a finite set of labels of instructions of P,
- $q_0 \in Q$ is the initial label,
- $h \in Q$ is the halting label, and
- P is the set of instructions of the following forms:

1. $p : (\mathrm{ADD}(k), q, s)$, with $p, q, s \in Q$, $1 \leq k \leq d$ ("increment"-instruction). Add 1 to register k and go to one of the instructions with labels q, s.
2. $p : (\mathrm{SUB}(k), q, s)$, with $p, q, s \in Q$, $1 \leq k \leq d$ ("decrement"-instruction). Subtract 1 from the positive value of register k and go to the instruction with label q, otherwise (if it is zero) go to the instruction with label s.

3. h : HALT (the halt instruction). Stop the computation of the machine.

For generating languages over T, we use the model of a *register machine with output tape* (introduced in [52], see also [5]), which also uses a tape operation:

4. $p : (\text{WRITE}(A), q)$, with $p, q \in Q$, $A \in T$.

The configuration of a register machine is given by the $(d+1)$-tuple $(q, n_1, \ldots, n_d)$, where $q \in Q$ and $n_i \geq 0, 1 \leq i \leq d$, describing the current label of the machine as well as the contents of all registers. A transition of the register machine consists in updating/checking the value of a register according to an instruction of one of types above and by changing the current label to another one. We say that the machine stops if it reaches the label h. A (non-deterministic) register machine $\mathcal{M}$ is said to generate a vector $(n_1, \ldots, n_m)$ of natural numbers if, starting from configuration $(q_0, 0, \ldots, 0)$ the machine stops in configuration $(h, n_1, \ldots, n_m, 0, \ldots, 0)$. The set of all vectors generated in this way by $\mathcal{M}$ is denoted by $Ps(\mathcal{M})$. It is known (e.g., see [52], [68]) that register machines generate $PsRE$. If the WRITE instruction is used, then RE can be generated.

In the case when a register machine cannot check whether a register is empty we say that it is *partially blind*; the second type of instructions is then written as $p : (\text{SUB}(k), q)$ and the transition is undefined if register k is zero.

Partially blind register machines have an implicit test for zero at the end of a (successful) computation: counters $m+1, \ldots, d$ should be empty. It is known [22] that partially blind register machines generate exactly $PsMAT^{\lambda}$ (Parikh sets of languages of matrix grammars without appearance checking).

2.2 Graph-controlled systems

Now we introduce the graph-controlled scheme that permits the construction of systems controlled by a graph.

Definition 2.2.1. Let V be a finite alphabet and $op : V^* \to 2^{V^*}$ be an arbitrary string substitution on V. We call op an *operation*.

Example 2.2.2. *Consider the following (string rewriting) operation* op_1 *which is given by the rule* $A \to BC$ *and consider the string* AAB*. Then* $op_1(AAB) = \{BCAB, ABCB\}$.

If there is no confusion we will not distinguish the operation and its finite representation (by rewriting or insertion-deletion rules).

Definition 2.2.3. Let $OP = \{op_1, \dots, op_l\}$, where $op_i, i \in \{1 \dots l\}$ is an operation. We denote by $Appl : OP \times V^* \to \{TRUE, FALSE\}$ the predicate that checks the applicability of an operation. $Appl(op, w)$ returns true if op is applicable to the word $w \in V^*$, and false otherwise.

Definition 2.2.4. A *graph-controlled* scheme is a tuple $\Pi = (V, T, A, i_0, i_f, R)$, where V is a (working) alphabet, $T \subseteq V$ is a terminal alphabet, $A \subset V^*$ is a finite set of axioms, $i_0 \in \{1, \dots, n\}, n = Card(R)$ is the initial label, $i_f \in \{1, \dots, n\}$ is the final label, R is a set of rules of the following form (i, op_i, P_i, F_i), where

- $i \in \{1, \dots, n\}$ is a label for the rule (unique for each rule),
- op_i, is a string rewriting operation, and
- $P_i, F_i \subseteq \{1, \dots, n\}$. Sets P_i and F_i are called *success* and *failure* fields correspondingly.

We note that for different indexes i, i' the operations op_i and $op_{i'}$ are not necessary distinct.

The *configuration* of a graph-controlled scheme Π is written as a pair (i, w), where $w \in V^*$ and $i \in \{1, \dots, n\}$ is the index of the rule to be applied. A derivation step $(i, w) \Rrightarrow (i', w')$ is performed if one of the following conditions hold:

- $Appl(op_i, w) = true, w' \in op_i(w)$, and $i' \in P_i$,
- $Appl(op_i, w) = false, w = w'$, and $i' \in F_i$.

If $F_i = \emptyset$ for every $i \in \{1, \dots n\}$ then such scheme is called graph-controlled scheme *without appearance checking*. Otherwise, it is called graph-controlled scheme *with appearance checking*. When it is not explicitly mentioned we consider graph-controlled schemes without appearance checking.

As usual the transitive and reflexive closure of $\Rrightarrow$ is denoted as $\Rrightarrow^*$. The language generated by graph-controlled scheme Π is defined as follows

$$L(\Pi) = \{w \in T^* \mid (i_0, x) \Rrightarrow^* (i_f, w), x \in A\}.$$

We give an alternative definition of the graph-controlled scheme.

Definition 2.2.5. A *graph-controlled* scheme Π is given by a tuple $(V, T, A, i_0, i_f, R_1, \dots, R_n)$, where elements V, T, A, i_0 and i_f are defined as for the graph-controlled scheme above. The set of rules forms a partition $R = R_1 \cup \dots \cup R_n$, where each R_j is called *component*. Each rule from R_i has the form $(op_{i,k}; p_{i,k}, f_{i,k})$, where

- $i \in \{1, \dots, n\}$ refers to label of the component, $k \in \{1, \dots, Card(R_i)\}$ is a distinct index of rule in i-th component,
- $op_{i,k}$ is an operation,
- $p_{i,k}, f_{i,k} \in \{1, ..., n\}$, $p_{i,k}$ and $f_{i,k}$ are called *success* and *failure* labels correspondingly.

For a configuration (i, w) of Π we say that the component i is active. A derivation step $(i, w) \Rrightarrow (i', w')$ is performed if one of following conditions hold:

- there is $k \in \{1, ..., n\}$, such that $Appl(op_{i,k}, w) = true, w' \in op_{i,k}(w)$, and $i' = p_{i,k}$,
- for all $k \in \{1, ..., n\}$ $\;Appl(op_{i,k}, w) = false, w = w'$ and $i' = f_{i,k'}$, for some $k' \in \{1, ..., n\}$.

The language generated by such a scheme is defined as

$$L(\Pi) = \{w \in T^* \mid (i_0, x) \Rrightarrow^* (i_f, w), x \in A\}.$$

Is is easy to see that the second definition of graph-controlled scheme can be easily transformed to the first one. The converse inclusion is also true and can be obtained by a subset construction. In what follows we shall consider the graph-controlled scheme defined as in the second definition.

We define the communication graph of a graph-controlled scheme as a graph with nodes $1, \ldots, n$ having an edge between node i and j if there exists a rule $(op_{i,k}; p_{i,k}, f_{i,k}) \in R_i$ and either $p_{i,k} = j$ or $f_{i,k} = j$. We are particularly interested in schemes whose communication graph has a tree structure.

If not stated otherwise we consider the systems without the appearance checking mechanism, *i.e.*, every failure label from $i-$th component is equal to i. In this case we omit $f_{i,k}$ in the definitions of rules.

Let us consider the following example.

Example 2.2.6. *Let $T = \{a\}$ be a terminal alphabet and $V = T \cup \{A, A'\}$ be a working alphabet. Let operations of the scheme be string rewriting operations. Appl(Op, w) is true, iff the left hand side of Op is present in w. Consider the following graph-controlled scheme (defined in the sense of definition 2.2.5):*

$$\Pi = (V, T, \{A\}, 0, 2, R_0, R_1, R_2\}),$$

where

$$R_0 = \{r_{0.1} : (A \to A'A'; 0, 1), \qquad r_{0.2} : (A \to a; 2, 1)\},$$
$$R_1 = \{r_1 : (A' \to A; 1, 0)\} \qquad R_2 = \{r_2 : (A \to a; 2, 2)\}.$$

The communication graph of Π is depicted on Figure 2.1. We claim that system Π generates language $\{a^{2^n} \mid n \geq 0\}$.

Rules $(Op_{0,1}; p_{0,1}, f_{0,1}), (Op_{0,2}; p_{0,2}, f_{0,2})$, $(Op_{1,1}; p_{1,1}, f_{1,1})$, and $(Op_{2,1}; p_{2,1}, f_{2,1})$ of the definition 2.2.5 correspond to the rules $r_{0.1}, r_{0.2}$, r_1, and r_2 of Π.

The rule $r_{0,1}$ is applicable as far as there is at least one nonterminal A presents in the string. Its application will rewrite this nonterminal A by $A'A'$. The rule $r_{0.1}$ is also applicable as far as there is a symbol A in the string. However, its action is different – A is rewritten to a and the string will be processed by component 2. When the configuration $(0, w)$ with $|w|_A = 0$ is reached, the string will be processed by component 1 (we also say that it is sent to component 1) because of the failure labels $f_{0,1} = 1$ and $f_{0,2} = 1$.

Starting from axiom A in configuration $(0, A)$ we can apply either rule $r_{0.1}$ or $r_{0.2}$ and get either configuration $(0, A'A')$ or $(2, a)$, correspondingly. In the latter case we produce $a \in T^*$ in the final component 2. Hence, $a \in L(\Pi)$.

In configuration $(0, A'A')$ we can only apply rule r_1 and get $(1, AA')$ or $(1, A'A)$. By applying r_1 once more to AA' (or $A'A$) we get the configuration $(1, AA)$. Now by failure label $f_{1,1} = 0$ the rule r_1 sends the string AA to component 0. By applying $r_{0.1}$ we get $(0, A'A'A)$ or $(0, AA'A')$ and by applying $r_{0.2}$ we get $(2, aA)$. In the latter case we can apply one time rule r_2 and get $(2, aa)$. Hence $aa \in L(\Pi.)$

By applying $r_{0.1}$ to $(0, A'A'A)$ (or $(0, AA'A')$) we get $(0, A'A'A'A')$ and $A'A'A'A'$ will be sent to component 1 by failure label $p_{0,1} = 1$ or $p_{0,2} = 1$. In case $A'A'A$ is replaced by $A'A'a$ (by rule $r_{0.2}$) such string is not terminal, and no rules can be further applied. Hence this computation can be omitted from consideration. From configuration $(1, A'A'A'A')$ in four steps we get $(1, AAAA)$ and the result is sent back to component 0.

In general from configuration $(0, A^n)$ in $n+1$ steps we get configuration $(1, A'^{2n})$, and then in $2n+1$ steps we get configuration $(0, A^{2n})$. Hence, we double the number of A per cycle that corresponds to rewriting productions $A \to A'A'$ and $A' \to A$ in components 0 and 1. By induction on the number of cycles we get that in component 0 appear all the strings from $\{A^{2^n-m}A'^{2m} \mid n \geq 0, 0 \leq m \leq 2^n\}$, and n is the number of the cycles.

Production $r_{0.2}$ terminates this cycle and only if the string does not contain nonterminals A' such a string produce terminal string in $L(\Pi)$.

Hence, in component 2 will appear the following strings

$$\{A^{2^n-m-l}A'^{2m}a^l \mid n \geq 0, 1 \leq m + l \leq 2^n, l \geq 1\}.$$

Considering all terminal strings in component 2, we get our language $L(\Pi) = \{a^{2^n} \mid n \geq 0\}$.

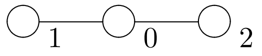


Figure 2.1: Graph structure for Example 2.2.6.

Chapter 3

Insertion systems

In this chapter we consider systems having only insertion rules. We use an inverse morphism and a weak coding as specific squeezing mechanisms that filter only those words of a language that have a "proper" structure. We consider both pure insertion and graph-controlled insertion systems. One of the main results of this chapter states the equivalence of generating power between context-free grammars and insertion systems of size $(3, 1, 1)$ (when obtained languages are encoded by the means of morphisms). Moreover, in a similar way it is shown an equivalence for the class of matrix grammars and graph-controlled insertion systems of size $(3, 1, 1)$. Another important theorem of the chapter proves the equality of graph-controlled insertion systems having size $(2, 2, 2)$ to the family of recursively enumerable languages.

3.1 Definitions

An *insertion system* is a construct $I = (V, A, R)$, where V is an alphabet, A is a finite language over V, and R is a finite sets of triples of the form (u, α, v), where u, α, and v are strings over V, $\alpha \neq \varepsilon$. The elements of V are working symbols, those of A are *axioms*, the triples in R are *insertion rules*. An insertion rule $(u, \alpha, v) \in R$ indicates that the string α can be inserted between u and v. Stated otherwise, $(u, \alpha, v) \in R$ corresponds to the rewriting rule $uv \to u\alpha v$. We denote by $\Rightarrow$ the relation defined by an insertion rule. Formally, $x \Rightarrow y$ iff $x = x_1uvx_2, y = x_1u\alpha vx_2$, for some $(u, \alpha, v) \in I$ and $x_1, x_2 \in V^*$. We denote by $\Longrightarrow^*$ the reflexive and transitive closure of $\Rightarrow$, and $\Rightarrow^+$ denote its transitive closure.

The language generated by I is defined by

$$L(I) = \{w \in V^* \mid x \Rightarrow^* w, x \in A\}.$$

The complexity of an insertion system $I = (V, A, R)$ is described by the vector (n, m, m') called *size*, where

$$n = \max\{|\alpha| \mid (u, \alpha, v) \in R\},$$
$$m = \max\{|u| \mid (u, \alpha, v) \in R\},$$
$$m' = \max\{|v| \mid (u, \alpha, v) \in R\}.$$

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

If some of the parameters n, m, m' is not specified, then we write instead the symbol $*$. In particular, $INS_*^{0,0}$ denotes the family of languages generated by insertion systems with rules having no contexts.

A *graph-controlled insertion system* is the graph-controlled scheme $\Pi = (V, A, i_0, i_f, R_1, \dots, R_n)$ (see definition 2.2.5), where for each rule (op, p, f) from $R_i, 1 \le i \le n$ the operation op is an insertion rule. By default, $Appl(op, w)$ is true iff the insertion rule $op = (u, \alpha, v)$ can be performed on w, *i.e.*, w contains a proper substring uv.

We remark that in the case of insertion systems there is no distinction between terminal and nonterminal alphabets.

We denote by $LSP_k(ins_n^{m,m'})$ the family of languages generated by graph-controlled insertion systems with $k \ge 1$ components and insertion rules of size at most (n, m, m') and whose communication graph has a tree structure. The letter t is inserted before P to denote classes whose communication graph is arbitrary, e.g., $LStP_k(ins_n^{m,m'})$. $LSP_k(ins_n^{m,m'})^{ac}$ denotes the family of languages generated by graph-controlled insertion systems having the size (n, m, m'), and having computation *with* appearance checking.

We remark that the definition of graph-controlled insertion systems almost coincides with the definition of insertion P systems [60]. In Chapter 5 we discuss the difference between these two models. Traditionally, in the literature, the term of insertion P systems is used for graph-controlled insertion systems, however, in what follows, we will use the latter term, because of a much simpler definition.

Now we give some examples of insertion and graph-controlled insertion systems.

Example 3.1.1. *Let* $I_1 = (\{a, b, c\}, \{abc\}, I)$*, where*

$$I = \{(a, a, \varepsilon), \qquad (b, b, \varepsilon), \qquad (c, c, \varepsilon)\}.$$

Clearly, this system generates the regular language $L(I_1) = \{a^+b^+c^+\}$*. Indeed, the axiom* $abc \in L(I_1)$ *and the insertion rules can insert an arbitrary number of* a, b *and* c *as long as there is a corresponding letter to the left.*

It is possible to consider the above example as a graph-controlled insertion system with three components with single rule per component, and where the next active component is determined by a cyclic order. Then the non context-free language $L_2 = \{a^n b^n c^n \mid n \geq 1\}$ is generated.

Example 3.1.2. *Consider the following graph-controlled insertion system* $\Pi_2 = (\{a, b, c\}, \{abc\}, 0, 0, R_0, R_1, R_2)$*, where*

$$R_0 = \{r_0 : (a, a, \varepsilon; 1)\};$$
$$R_1 = \{r_1 : (b, b, \varepsilon; 2)\};$$
$$R_2 = \{r_2 : (c, c, \varepsilon; 0)\}.$$

Clearly, $\Pi_2 \in LStP_3(ins_1^{1,0})$ *and* $abc \in L(\Pi_2)$*. The insertions of* a, b *and* c *are performed when the components 0,1 and 2 are active, correspondingly. This implies that the number of* a, b *and* c *when component* 0 *is active is the same. Hence,* $L(\Pi_2) = \{a^n b^n c^n \mid n \geq 1\}$*.*

3.2 Computational power of pure insertion systems

It is known that the classes of languages obtained by systems using only insertions, are incomparable with many known language classes. As an example consider the linear language $\{a^n b a^n \mid n \geq 1\}$. This language cannot be generated by any insertion system (see Theorem 6.6 in [62]). In order to overcome this "obstacle" we use some codings to interpret the generated strings. More precisely, firstly an inverse morphism and then a weak coding are applied to the generated string. Hence, we consider the languages of the form: $\varphi(h^{-1}(L(\Pi)))$, where Π is an insertion system, φ is a weak coding and h is a morphism.

We note that in the literature there were also considered another types of codings when an intersection with a (regular) language is applied to the results of the insertion system instead of the inverse morphism (see, for example, [56, 61]). We mention that these types of codings are rather simple and can be simulated by a finite state transducer. Using this method we show several characterizations of language classes from the Chomsky hierarchy in terms of insertion systems.

We start with the following example where it is shown that a non-regular context-free language can be generated by an insertion system of size $(1,1,0)$ without any coding.

Example 3.2.1. *Consider a system $I = (T, \{a\}, R)$, where $T = \{a,b,c,d\}$ and R is defined as follows: $R = \{(a,b,\varepsilon),(b,c,\varepsilon),(c,d,\varepsilon),(d,a,\varepsilon)\}$.*

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

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

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

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

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

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

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

Next theorem is from [62].

Theorem 3.2.2. $INS_*^{1,1} \subseteq CF$.

Proof. For an insertion system $\Pi = (T, A, I)$ consider a context-free grammar $G = (N, T, S, P)$ having nonterminal alphabet $N = \{D_{a,b} \mid a, b \in T \cup \{\varepsilon\}\}$ and the set of productions $P = P_1 \cup P_2 \cup P_3$, where

$$\begin{aligned} P_1 = &\quad \{S \to \delta(\varepsilon, w, \varepsilon) \mid w \in A\}, \\ P_2 = &\quad \{D_{a,b} \to a \mid D_{a,b} \in N, a, b \in T \cup \{\varepsilon\}\}, \\ P_3 = &\quad \{D_{\overline{a_1},\overline{a_2}} \to \delta(\overline{a_1}, w, \overline{a_2}) \mid (a_1, w, a_2) \in I, \\ &\quad \text{for } l = 1,2\ \ \overline{a_l} = a_l, \text{ if } a_l \neq \varepsilon \text{ and } \overline{a_l} \in T \cup \{\varepsilon\}, \text{ if } a_l = \varepsilon\}, \end{aligned}$$

where for every $a_1, a_2 \in T \cup \{\varepsilon\}, w \in T^*$ we denote by $\delta(a_1, w, a_2)$ the following function

$$\delta(a_1, w, a_2) = \begin{cases} D_{a_1,a_2}, & \text{if } w = \varepsilon \\ D_{a_1,b_1}D_{b_1,b_2}\dots D_{b_{k-1},b_k}D_{b_k,a_2}, & \text{if } w = b_1\dots b_k. \end{cases}$$

The rule $(a_1, b_1 \dots b_k, a_2) \in I, a_1, a_2 \in T, b_1 \dots b_k \in T^k$ can be simulated by the grammar iff the corresponding sentential form contains D_{a_1,a_2}. It is clear that non-terminals in D preserve one symbol left and right contexts. Hence, there is a following derivation $wD_{a_1,a_2}w' \Rightarrow wD_{a_1,b_1}D_{b_1,b_2}\dots D_{b_{k-1},b_k}D_{b_k,a_2}w'$. Since index symbols are duplicated in adjacent nonterminals we have one symbol context being preserved in the resulted string. The simulation of rules that have no contexts is performed by productions from P_3 with arbitrary contexts: $\overline{a_l} \in T \cup \{\varepsilon\}, l = 1, 2$.

The simulation starts by the production

$$S \to D_{\varepsilon,b_1} D_{b_1,b_2} \dots D_{b_{k-1},b_k} D_{b_k,\varepsilon} \in P_1$$

corresponding to the choice of an axiom from A. The terminal string is obtained by applying rules of P_2 at the end of derivation. Hence, we have shown that each step of derivation in Π can be reproduced by the context-free grammar G.

The other direction $L(G) \subseteq L(\Pi)$ can be shown as follows. A derivation in G starts from production from P_1 $S \to \delta(\varepsilon, w, \varepsilon)$, where $\delta(\varepsilon, w, \varepsilon) = D_{\varepsilon,b_1} D_{b_1,b_2} \dots D_{b_{k-1},b_k} D_{b_k,\varepsilon}$, and $w = b_1 b_2 \dots b_k \in A$. This corresponds to choosing the axiom w in a derivation of Π. Then either productions from P_2 or P_3 can be applied. Each rule $D_{a,b} \to a \in P_2$ rewrites $D_{a,b}$ by terminal a. In the corresponding derivation of Π it means that no more insertions can be done between a and b using a as a left context. (Clearly, for every derivation in G one may consider an equivalent derivations in which the rules from P_2 are applied an the end.)

Each rule $D_{\overline{a_1},\overline{a_2}} \to \delta(\overline{a_1}, w, \overline{a_2}) \in P_3$ corresponds to insertion of w between $\overline{a_1}$ and $\overline{a_2}$ in the derivation of Π. This captures also the case when the insertion rule has empty left and/or right contexts.

Since there is a one to one correspondence between derivations in G and Π we obtain $L(\Pi) = L(G)$. Hence, $INS_*^{1,1} \subseteq CF$. □

Next we give a characterization of context-free languages by means of insertion systems of size $(3, 1, 1)$.

Theorem 3.2.3. *A language L is context-free if and only if it can be represented in the form $L = \varphi(h^{-1}(L'))$ where $L' \in INS_3^{1,1}$, φ is a weak coding and h is a morphism.*

Proof. Taking into account Theorem 3.2.2 and the closure property of context-free languages with respect to inverse morphisms and weak codings we get the "only if" part of the statement.

In order to prove the "if" part of the theorem, it is enough to show that for any context-free language L there is an insertion system Π of size $(3,1,1)$, such that $L = \varphi(h^{-1}(L(\Pi)))$, where h is a morphism, and φ is a weak coding.

Let $G = (N, T, S, P)$ be a context-free grammar in the Chomsky normal form such that $L = L(G)$. Consider the following insertion system $\Pi = (V, \{S\$\}, I)$, where $V = T \cup N \cup \{\#, \$\}$, $I = \{(A, \#\gamma, \alpha) \mid \alpha \in V\backslash\{\#\},\ A \to \gamma \in P, \gamma \in (T \cup V)^*, 1 \leq |\gamma| \leq 2\}$. The morphism h and the weak coding φ are defined as follows

$$h(a) = \begin{cases} a\#, & \text{if } a \in V\backslash(T \cup \{\#\}), \\ a, & \text{if } a \in T \cup \{\$\}, \end{cases} \qquad \varphi(a) = \begin{cases} a, & \text{if } a \in T, \\ \varepsilon, & \text{if } a \in V\backslash T. \end{cases}$$

We claim that $L(\Pi) = L(G)$. Indeed, each rule $(A, \#\gamma, \alpha) \in R$ can be applied to the sentential form $wA\alpha w'$ if and only if $\alpha \neq \#$. Hence, a production $A \to \gamma \in P$ can be simulated by the corresponding rule in Π. For the convenience we add a special symbol \$ as the right border.

When every nonterminal is marked and no rules can be applied the output word can be subjected to the inverse morphism h^{-1}. Indeed, if the system produces a word having some unmarked nonterminal then h^{-1} is not defined. At this point h^{-1} removes all marking symbols, and φ removes all nonterminal symbols. This proves the assertion of the theorem. □

Since Parikh image of a context-free language is always semilinear, this implies that insertion systems of size $(n,1,1)$ can generate only languages whose Parikh image is semilinear. If we drop the contexts then we have a strict inclusion:

Lemma 3.2.4. $Ps(INS_*^{0,0}) \subset SL$.

Proof. Consider the following semilinear set

$$P = \{(0,0) + \sum_{k \geq 0} k(0,1)\} \cup \{(0,0) + \sum_{k \geq 0} k(1,0)\}.$$

We claim that there is no insertion system whose Parikh set is equal to P. We shall prove this statement by contradiction. Assume that there is an insertion system γ such that $Ps(L(\gamma)) = P$. Let γ have the terminal alphabet $\{a, b\}$. Since the language of axioms is finite the system contains some insertion rules of the form $(\varepsilon, a^k, \varepsilon)$, and

$(\varepsilon, b^l, \varepsilon), k, l \geq 1$. Then by applying both rules in a derivation we thus obtain a word with both letters a and b. This is a contradiction. □

On the other hand every linear set can be generated by context-free insertion systems.

Lemma 3.2.5. *For every linear set S there is a language $L \in INS_*^{0,0}$ such that $S = Ps(L)$.*

3.3 Graph-controlled insertion systems

In the remaining of the chapter we consider graph-controlled insertion systems. We show that these systems have more computational power than pure insertion systems.

Theorem 3.3.1. $LStP_*(ins_*^{1,1}) \subset MAT^\lambda$.

Proof. In order to prove the statement it is enough to show that for any graph-controlled insertion system Π there is a matrix grammar G such that $L(\Pi) = L(G)$. We extend the construction of the context-free grammar used in Theorem 3.2.3 for the case of matrix grammars.

Let $\Pi = (V, A, i_0, i_f, R_1, \dots, R_n)$ be an arbitrary graph-controlled insertion system. Consider the matrix grammar $G = (N, V, S, M)$ having nonterminal alphabet $N = Q \cup D$, where $Q = \{Q_i \mid i = 1, \dots, n\}, D = \{D_{a,b} \mid a, b \in V \cup \{\varepsilon\}\}$, and the set of matrices $M = M_1 \cup M_2 \cup M_3$, where

$$\begin{aligned}
M_1 =& \{(S \to Q_{i_0}\delta(\varepsilon, w, \varepsilon)) \mid w \in A\},\\
M_2 =& \{(D_{a,b} \to a) \mid D_{a,b} \in D, a, b \in T \cup \{\varepsilon\}\} \cup \{(Q_{i_f} \to \varepsilon)\},\\
M_3 =& \{(Q_i \to Q_j, D_{\overline{a_1},\overline{a_2}} \to \delta(\overline{a_1}, w, \overline{a_2})) \mid (a_1, w, a_2; j) \in R_i,\\
& \quad \text{for } l = 1,2 \;\; \overline{a_l} = \begin{cases} a_l, & \text{if } a_l \in V,\\ t, \forall t \in V \cup \{\varepsilon\}, & \text{if } a_l = \varepsilon \end{cases} \quad \},
\end{aligned}$$

where for every $a_1, a_2 \in V \cup \{\varepsilon\}, w \in V^*$ we denote by $\delta(a_1, w, a_2)$ the following

$$\delta(a_1, \varepsilon, a_2) = D_{a_1,a_2},\ \delta(a_1, b_1 \dots b_k, a_2) = D_{a_1,b_1} D_{b_1,b_2} \dots D_{b_{k-1},b_k} D_{b_k,a_2}.$$

The simulation of Π by the matrix grammar G is based on the encoding of pairs of adjacent letters by nonterminals from D. So, the encoded pair can be used as a context for an insertion rule. In addition, the label of the active component is represented by a nonterminal in Q. A rule $(a_1, b_1 \dots b_k, a_2; j) \in R_i, a_1, a_2 \in V \cup \{\varepsilon\}, b_1 \dots b_k \in V^k$ can be simulated by the grammar iff the sentential form contains both Q_i and D_{a_1,a_2}. As a result, the label representing the active component is rewritten to Q_j and D_{a_1,a_2} is rewritten to the string $D_{a_1,b_1}D_{b_1,b_2} \dots D_{b_{k-1},b_k}D_{b_k,a_2}$. It is clear that the string preserves one symbol (left) context. In order to simulate rules that have no contexts we introduce productions with an arbitrary contexts: $\overline{a_l} \in V \cup \{\varepsilon\}, l = 1, 2$.

The simulation of Π by the grammar starts with a nondeterministic choice of an axiom from A. Then, during the derivation each rule from $R_1, \dots, R_n$ having the context (a_1, a_2) can be applied iff the productions having D_{a_1,a_2} in the left hand side can be applied. Finally, the string over V can be produced by the grammar as soon as Q_{i_f} is deleted from the sentential form. The deletion of Q_{i_f} specifies that Π activates the final component. As there is one to one correspondence between derivations in G and Π we obtain $L(\Pi) = L(G)$. Hence, $LStP_*(ins_*^{1,1}) \subseteq MAT^\lambda$.

The strictness of the inclusion follows from the fact there are languages from MAT^λ which cannot be generated by any insertion P system from $LStP_m(ins_n^{1,1})$, for any $n \geq 1$. Indeed, consider the context-free language $L_a = \{ca^kca^kc \mid k \geq 0\}$. Since every context-free language is a matrix language [63] we have $L_a \in MAT^\lambda$. On the other hand, $L_a \notin LStP_m(ins_n^{1,1})$, for any $n \geq 1$. For the contrary, assume there is such a system Π'. We note, that the system cannot delete or rewrite any letter, so every insertion is terminal. As the languages of axioms are finite we need an insertion rule of letter a. Consider the final insertion step in a derivation which has at most one step and derives a word $w = ca^kca^kc$, for some $k \geq n + 1$:

$$w_0 \Rightarrow^* w' \Rightarrow w,$$

where w_0 is an axiom. Since $|w_0|_c \leq 3$, c may be inserted by the last insertion. Assume, that $|w'|_c = 3$. In the latter case, let a^p be the inserted string, $p \leq n$. Because, we may insert a^p in the distinct positions of w' we get that either $ca^{k-p}ca^{k+p}c \in L(\Pi')$ or $ca^{k+p}ca^{k-p}c \in L(\Pi')$. This is a contradiction.

Now assume that c is inserted by the last insertion. We note that the insertion of two c is not possible, since $k \geq n+1$. Consider three cases: (1) the last applied rule inserts c in the middle, (2) at the end, or (3) at the beginning of w'.
(1) Let $w_c = a^{p'}ca^{p''}$ be the inserted string, where $p'+p'' \leq n-1$. Hence, $w' = ca^{k'+k''}c$, where $k'+p' = k''+p'' = k$, and $k'+k'' = 2k-p'-p'' \geq 2n+2-n+1 \geq 4$. Obviously, regardless of the contexts of the last insertion rule there are at least two positions at which w_c can be inserted. So, we get a contradiction because either $ca^{k'+p'+1}ca^{k''+p''-1}c \in L(\Pi')$, or $ca^{k'+p'-1}ca^{k''+p''+1}c \in L(\Pi')$.
(2) Let a^qc be the inserted string, where $q \leq n-1$. The corresponding insertion rule has one of the following forms: $(\varepsilon, a^qc, \varepsilon; j)$ or $(a, a^qc; \varepsilon, j)$, where j is an index of the final component. In ether case, a^qc may be inserted in w' before the last letter a. This is a contradiction. The case (3) is a mirror to the case (2) and can be treated similarly.

So we proved $L_a \notin LStP_n(ins_2^{1,1})$, for any $n \geq 1$ and, hence, $LStP_*(ins_*^{1,1}) \subset MAT^\lambda$. □

Since a tree is a special case of a graph we get the following result

Corollary 3.3.2. $LSP_*(ins_*^{1,1}) \subset MAT^\lambda$.

Let us consider graph-controlled insertion systems with left and right contexts of at most one symbol. This family can characterize the languages generated by context-free matrix grammars, if a specific squeezing mechanism is used.

Theorem 3.3.3. *A language L is in MAT^λ if and only if it can be written in the form $L = \varphi(h^{-1}(L'))$, where $L' \in LSP_*(ins_2^{1,1})$, φ is a weak coding, h is a morphism.*

Proof. Taking into account Theorem 3.3.1 and the fact that the class of languages generated by context-free matrix grammars is closed under inverse morphisms and weak codings we get a characterization of MAT^λ if we show that for every $L \in MAT^\lambda$ there is a weak codings φ, a morphism h, and a system Π such that $L(\Pi) \in LSP_*(ins_2^{1,1})$ and $L = \varphi(h^{-1}(L(\Pi)))$.

Let us consider a language $L \in MAT^\lambda$. Let $G = (N, T, S, M)$ be a matrix grammar in the binary normal form such that $L = L(G)$. We assume that matrices in M are labeled by integers $1,\dots,n$ and each matrix in M has the form $i:(A\to BC, A'\to B'C')$, where $A,A'\in N$ and $B,C,B',C'\in N\cup T\cup\{\varepsilon\}$. Consider the following graph-controlled insertion system Π with nonterminal alphabet $V=N\cup T\cup\{\#,\$\}\cup\{C_i,C_i'\mid i=1\dots n\}$, the initial and the final component labeled by "1", initial string $S\$$, and the communication graph having the structure represented in Figure 3.1

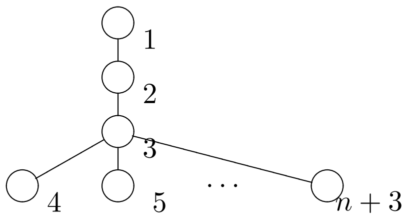


Figure 3.1: Communication graph for Theorem 3.3.3.

Let $i:(A\to BC, A'\to B'C')$ be a matrix in M. Then consider the sets of rules of the size $(2,1,1)$ that correspond to i-th production:

$$\begin{aligned}
R_1^i &=\{r_i.1.1:(A,\#C_i,\alpha;2) && \mid \alpha\in V\backslash\{\#\}\};\\
R_2^i &=\{r_i.2.1:(C_i,BC,\alpha;3), \quad r_i.2.2:(C_i',\#,\alpha;1) && \mid \alpha\in V\backslash\{\#\}\};\\
R_3^i &=\{r_i.3.1:(C_i,\#,\alpha;i+3), \quad r_i.3.2:(C_i',B'C',\alpha;2) && \mid \alpha\in V\backslash\{\#\}\};\\
R_{i+3} &=\{r_{i+3}.4:(A',\#C_i',\alpha;3) && \mid \alpha\in V\backslash\{\#\}\}.
\end{aligned}$$

We associate with k-th component $k=1,2,3$ the set of rules $R_k=\cup_{i=1\dots n}R_k^i$, and with k'-th component $k'=4\dots n+3$ the set $R_{k'}$.

Let h and φ be a morphism and a weak coding defined as follows:

$$h(a)=\begin{cases} a, & \text{if } a\in T\cup\{\$\},\\ a\# & \text{if } a\in V\backslash(T\cup\{\#\})\end{cases}\qquad \varphi(a)=\begin{cases} a, & \text{if } a\in T,\\ \varepsilon & \text{if } a\in V\backslash T.\end{cases}$$

We claim, $L(G)=\varphi(h^{-1}(L(\Pi)))$. Indeed, Π simulates productions of M in a direct way. Every sentential form contains at most one unmarked symbol from $\{C_i,C_i'\mid i=1\dots n\}$. Whenever the rule $i.1.1$ is applied, the only possible derivation is to complete all the rules corresponding to i-th production. Consider sentential form $w_1AwA'w_2$, where $w_1,w_2,w\in V^*$ and A,A' are not marked and suppose there is a matrix rule of the form $(A \to BC, A' \to B'C')$.

$$(1, w_1AwA'w_2) \Rrightarrow_{r_i.1.1} (2, w_1A\#C_iwA'w_2) \Rrightarrow_{r_i.2.1}$$
$$(3, w_1A\#C_iBCwA'w_2) \Rrightarrow_{r_i.3.1} (4, w_1'A'w_2) \Rrightarrow_{r_{n+i}.4} (3, w_1'A'\#C_i'w_2) \Rrightarrow_{r_i.3.2}$$
$$(2, w_1'A'\#C_i'B'C'w_2) \Rrightarrow_{r_i.4.2} (1, w_1'A'\#C_i'\#B'C'w_2),$$

where $w_1' = w_1A\#C_i\#BCw$. Hence, the derivation marks nonterminals A, A' and inserts BC, $B'C'$ to the right of $A\#$ and $B\#$, correspondingly. We note, that we add one symbol \$ to the right end in order to permit the contextual insertion for the rightmost nonterminal(s). At the end, by applying the inverse morphism and the weak coding we remove every marked nonterminal. Hence, we have $L(G) \subseteq \varphi(h^{-1}(L(\Pi)))$.

The inverse inclusion is obvious, because every rule in Π has its counterpart in G. Moreover the case when the derivation in Π is blocked corresponds to the case in which the simulation of a matrix cannot be completed. Hence, we get the $L(G) = \varphi(h^{-1}(L(\Pi)))$. □

Since trees are the special case of graphs we obtain the same result for graph-controlled systems with an arbitrary structure:

Corollary 3.3.4. *A language $L \in MAT^\lambda$ if it can be written in the form $L = \varphi(h^{-1}(L'))$, where $L' \in LStP_*(ins_2^{1,1})$, φ is a weak coding, h is a morphism.*

We mention that a similar result can be obtained with a smaller number of components but increasing the length of inserted words.

Lemma 3.3.5. *For any context-free matrix grammar G' there is a graph-controlled insertion system Π' such that $L(\Pi') \in LSP_{n+1}(ins_3^{1,1})$ and $L(G') = L(\Pi')$, where n is the number of matrices in G'.*

Proof. In order to prove the lemma we use the same argument as in the previous theorem. For a matrix $i : (A \to BC, A' \to B'C') \in M, i = 1, \dots, n$ we consider sets of rules

$$R_1'^i = \{(A, \#BC, \alpha; i + 1)\},$$
$$R_{i+1}' = \{(A', \#B'C', \alpha; 1)\}.$$

for every $\alpha \in V\backslash\{\#\}$.

Then, we replace components $R_1, \dots, R_{n+3}$ from Theorem 3.3.3 by $R'_1, R'_2, \dots, R'_{n+1}$, where $R'_1 = \cup_{i=1,\dots,n} R'^{i}_1$.

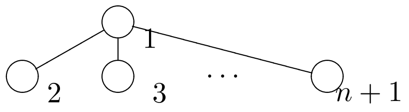


Figure 3.2: Communication graph for Lemma 3.3.5.

□

Taking into account that the class of matrix grammars with appearance checking equals RE we get the following corollary.

Corollary 3.3.6. *A language L is RE if and only if it can be written in the form $L = \varphi(h^{-1}(L'))$, where $L' \in LSP_*(ins_2^{1,1})^{ac}$, φ is a weak coding, h is a morphism.*

In order to prove the next theorem we use the "mark and migration" technique for insertion systems(see, e.g., [35, 62]). According to this technique, symbols that have been rewritten are marked (with the *marking symbols* # and $\overline{\#}$). We say that a letter a is *marked* in a sentential form waw' if it is followed by #, i.e., $|w'| > 0$, and # is the prefix of w'. For example, in order to simulate a context-free production $A \to BC$ the string $\#BC$ is inserted adjacent right to A, assuming that A is not yet marked. This can be done by the rule $(A, \#BC, \alpha), \alpha \in V \setminus \{\#\}$. As soon as the derivation of the simulated sentential form is completed, every nonterminal A must be marked and the pairs $A\#$ are subject to the inverse morphism.

In order to simulate a context sensitive production $AB \to AC$ we need to bring A beside B, because they can be separated by a string that consists of marked symbols. The migration of a symbol A over the marked context is performed stepwise by means of insertion of new nonterminal F_A to the right of the marked symbols. For example, assume we have to migrate A over $X\#$ in the string $AX\#\alpha$. We introduce the following insertion rules

$$(AX\#, F_A, \alpha),$$
$$(A, \#, X\#F_A\alpha),$$
$$(A\#X\#F_A, \#A, \alpha),$$

where $\alpha \in V \setminus \{F_A, \#\}$.

These rules perform the migration of A in the following derivation

$$AX\#\alpha \Rightarrow AX\#F_A\alpha \Rightarrow A\#X\#F_A\alpha \Rightarrow A\#X\#F_A\#A\alpha$$

One can check, that the migration shown above requires contexts of the insertion rules to be at least five symbols. The contexts of these rules check that the insertion rules can be performed in the above defined order. In the literature there are known results when this migration has been performed with rules of smaller size, e.g. the rules of size $(3, 3, 3)$ were considered in [35]. In order to further reduce the length of the contexts we use a graph-controlled insertion system. This allows the rules not to interfere with each other in the same part of the sentential form while migrating simultaneously two symbols. We consider two symbol contextual insertion rules and prove computational completeness for the graph-controlled insertion system with three components. In the next theorem we perform the migration of symbols to the *right*. Clearly, the same result can be achieved by migration symbols to the *left* if we consider symmetrical rules.

Theorem 3.3.7. *Every language $L \in RE$ can be represented in the form $L = \varphi(h^{-1}(L'))$, where φ is a weak coding, h is a morphism, and $L' \in LSP_3(ins_2^{2,2})$.*

Proof. According to the Church's thesis we need to prove only the inclusion of the family RE into the family of languages $\varphi(h^{-1}(LSP_3(ins_2^{2,2})))$. A simulation of a type-0 grammar in the special Pentonnen normal form is performed by means of "mark and migration" technique.

Let $G = (N, T, S, P)$ be a grammar in the special Pentonnen normal form. Consider a graph-controlled insertion system $\Pi = (V, \{S\$\}, i_0, i_f, R_1, R_2, R_3)$, where $V = T \cup N \cup F \cup \overline{F} \cup \{\#, \overline{\#}, \$\}$, $F = \{F_A, \mid A \in N\}$, $\overline{F} = \{\overline{F}_A, \mid A \in N\}$, and the initial and the final components are labeled by 1.

We assume that the communication graph of Π has the tree structure depicted in Figure 3.3.

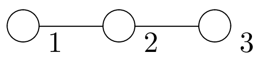


Figure 3.3: Communication graph for Theorem 3.3.7.

The sets of rules R_1, R_2, R_3 corresponding to the 1st, the 2nd, and the 3rd component are defined as follows:

$$\begin{aligned}
R_1 =&\{r_i.1.1 : (AB, \#C, \alpha; 1) \mid i : AB \to AC \in P, \alpha \in V\backslash\{\#, \overline{\#}\}\} \cup \\
&\{r_i.1.2 : (A, \#C, B\alpha; 1) \mid i : AB \to CB \in P, \alpha \in V\backslash\{\#, \overline{\#}\}\} \cup \\
&\{r_i.1.3 : (A, C, \alpha; 1) \mid i : A \to AC \in P, \alpha \in V\backslash\{\#, \overline{\#}\}\} \cup \\
&\{r_i.1.4 : (\varepsilon, C, A\alpha; 1) \mid i : A \to CA \in P, \alpha \in V\backslash\{\#, \overline{\#}\}\} \cup \\
&\{r_i.1.5 : (A, \#\delta, \alpha; 1) \mid i : A \to \delta \in P, \alpha \in V\backslash\{\#, \overline{\#}\}\}\cup \\
&\{r_A.1.6 : (A, \#F_A, \alpha; 2) \mid A \in N, F_A \in F, \alpha \in V\backslash\{\#, \overline{\#}\}\};
\end{aligned}$$

$$R_2 = \left\{ \begin{array}{l|l} r_A.2.1 : (F_A, \#A, \alpha'; 1) & A \in N, F_A \in F, \overline{F_A} \in \overline{F}, \\ r_A.2.2 : (\overline{F_A}, \overline{\#}A, \alpha'; 1) & \alpha' \in (N+T)\dot{(}N+T+\$) + \$ \end{array} \right\} \cup$$

$$\{r_A.2.3 : (F_A X, \#\overline{F_A}, \#; 3) \mid X \in F \cup N, \overline{F_B} \in \overline{F}, \alpha \in V\backslash\{\#, \overline{\#}\}\}\cup$$

$$\begin{array}{ll}
\{r_A.2.4 : (F_A\overline{F_B}, \overline{\#}F_A, \overline{\#}; 3), & r_A.2.5 : (F_A\overline{\#}, F_A, \alpha; 3), \\
r_A.2.6 : (\overline{F_A}\#, F_A, \alpha; 3), & r_A.2.7 : (F_A\overline{\#}, F_A, \overline{\#}; 3), \\
r_A.2.8 : (\overline{F_A}\#, F_A, \overline{\#}; 3), & r_A.2.9 : (F_A\overline{\#}, \overline{F_A}, \#; 3), \\
r_A.2.10 : (\overline{F_A}\#, \overline{F_A}, \#; 3)\}; &
\end{array}$$

$$\begin{aligned}
R_3 =&\{r_A.3.1 : (F_A, \#, \alpha; 2) \mid \alpha \in V, \alpha \neq \#\}\cup \\
&\{r_A.3.2 : (\overline{F_A}, \overline{\#}, \alpha'; 2) \mid \alpha' \in V, \alpha' \neq \overline{\#}\}.
\end{aligned}$$

Consider also the morphism $h : V\backslash\{\#\} \to V$ and the weak coding $\varphi : V \to T \cup \{\varepsilon\}$ defined by:

$$h(a) = \begin{cases} a, & \text{if } a \in T, \\ a\# & \text{if } a \in V\backslash(T \cup \{\#\}) \end{cases} \qquad \varphi(a) = \begin{cases} a, & \text{if } a \in T, \\ \varepsilon & \text{if } a \in V\backslash T. \end{cases}$$

One may see that each production in P has a one to one correspondence with an insertion rule from R_1. Furthermore, the insertions performed by rules $r_i.1.1 - r_i.1.5$ have the following properties:

- the rules can be only applied to the symbols that are not marked;
- the insertion marks the letter that is rewritten by the production.

Hence, for every derivation step in G a derivation step in Π can be considered (assuming that letters for context-sensitive production are not separated by marking symbols). The rule $r_i.1.6 : (A, \#F_A, \alpha; in_2)$, specifies that each unmarked letter from N may be subjected to the transfer.

Consider a pair of letters AB subjected to a production $AB \to AC$ or $AB \to CB \in P$. Suppose that this pair is separated by letters that have been marked. In this case the rules from the components R_2 and R_3 are used in order to transfer a copy of letter $A \in N$ to the right-hand side of marked symbols. Indeed, every rule from $r_i.2.3, \dots, r_i.2.10$ foresee the next symbol to the right and if it is marked, the rule inserts a copy of the symbol that have to be transferred to the right. We note, these rules make copied of the transfered symbol to the right in such a way that the inserted symbol would not be marked. In order to do so, the appropriate rule chooses to insert either the overlined copy $\overline{F_A}$ or the simple copy F_A. The rules $r_i.2.3, r_i.2.4$ describe the jump over one letter not in $\{\#, \overline{\#}\}$, and $r_i.2.5, \dots, r_i.2.10$ describe the jump over $\#$, $\overline{\#}$. Every rule $r_i.2.3, \dots, r_i.2.10$ sends the sentential form to the third component, and the rules $r_i.3.1, r_i.3.2$ in the third component send the sentential form back to the second component after marking one symbol $F_A \in F$ or $\overline{F_A} \in \overline{F}$.

The rules $r_i.2.1$ and $r_i.2.2$ may terminate the transferring procedure and send the sentential form to the first component if letter \$ or two letters from $\{ab \mid a \in N \cup T, b \in N \cup T \cup \{\$\}\}$ appear in the right context.

For example consider the transfer of A in the string $AX\#C\$$ (here, we underline the inserted substrings)

$$AX\#C\$ \overset{r.1.1}{\Rightarrow} A\ \underline{\#F_A}\ X\#C\$ \overset{r.2.3}{\Rightarrow} A\#F_AX\ \underline{\#\overline{F_A}}\ \#C\$ \overset{r.3.1}{\Rightarrow}$$
$$A\#F_A\ \underline{\#}\ X\#\overline{F_A}\#C\$ \overset{r.2.6}{\Rightarrow} A\#F_A\#X\#\overline{F_A}\ \underline{\#F_A}\ C\$ \overset{r.3.2}{\Rightarrow}$$
$$A\#F_A\#X\#\overline{F_A}\ \ \underline{\overline{\#}}\ \#F_AC\$ \overset{r.2.1}{\Rightarrow} A\#F_A\#X\#\overline{F_A}\ \overline{\#}\#F_A\ \underline{\#A}\ C\$.$$

We note that the 2-nd and the 3-rd components are activated in turns working in a cycle until either the rule $r_i.2.1$ or the rule $r_i.2.2$ is applied. In this case a copy of the symbol is inserted adjacent left to either an unmarked nonterminal, a terminal symbol, or the rightmost mark.

In order to to prove $\varphi(h^{-1}(L(\Pi))) = L(G)$ we observe that the sentential form preserves the following property: (i) The first component does not contain unmarked letters from $F \cup \overline{F}$; there is exactly one unmarked letter from $F \cup \overline{F}$ in the second component; and there are always two unmarked letters from $F \cup \overline{F}$ in the third component.

We mention that property (i) is preserved by every derivation. Indeed, we start derivation from the axiom $S\$$ that satisfies the property, then one unmarked symbol is inserted by $r_i.1.1$. Rules $r_i.2.3, \ldots, r_i.2.12$ always add one more unmarked letter, whereas rules $r_i.2.1, r_i.2.2, r_i.3.1, r_i.3.2$ always mark one letter from $F \cup \overline{F}$.

We consider only those words obtained in Π where every nonterminal symbol has been marked, because otherwise the inverse morphism h^{-1} is not defined. This implies that every cycle happening in the 2-nd and the 3-rd components must be terminated. Finally, by applying the weak coding φ we eliminate every nonterminal and marking symbols.

We also note that every reachable sentential form in G will be reachable also in Π by simulating the same production. Therefore, for every derivation in G one obtains a counterpart derivation in Π. This gives $L(G) = \varphi(h^{-1}(L(\Pi)))$. □

For any graph-controlled insertion systems with an empty axiom set the operation of *parallel merging* can be defined. The construction is similar to the one used for the union of finite automaton. Consider the system with every component from the original systems and two new components (for new initial and new final components). Assume the systems have disjoint sets of labels. If this is not the case then we can rename the labels. Then, add the identity rules $(\varepsilon, \varepsilon, \varepsilon; i_l)$ to the new initial component R_{i_0}, where i_l are the labels of the initial components for the original systems. Finally, for each finale component of the original system we add the identity rule $(\varepsilon, \varepsilon, \varepsilon; i_f)$, where i_f is the label of the new final component R_{i_f}.

The parallel merging has the property that the language generated by the merged system is equal to the union of languages of the systems being merged. Formally, let $\Pi^l = (V^l, \{\varepsilon\}, i_0^l, i_f^l, R_1^l, \ldots, R_n^l), l = 1, 2$, then let us denote by the merged system $\Pi = (V^1 \cup V^2, \{\varepsilon\}, i_0, i_f, R_{i_0}, R_{i_f}, R_1^1, \ldots, R_n^1, R_1^2, \ldots, R_n^2)$. Then we have $L(\Pi) = L(\Pi^1) \cup L(\Pi^2)$.

Now we consider the Parikh image of the graph-controlled insertion systems. An equivalence between the Parikh image of $LSP_*(ins_1^{0,0})$ and the family of semi-linear sets can be obtained:

Theorem 3.3.8. $SL = PsLStP_*(ins_1^{0,0})$.

Proof. Let $\Pi = (V, A, i_0, i_f, R_1, \ldots, R_n)$ be a graph-controlled insertion system such that $L(\Pi) \in LStP_*(ins_1^{0,0})$. Let $K = Ps(L)$. We may assume that i-th component has the form $R_i = \{r : (\varepsilon, x, \varepsilon; j)\}, x \in V, i = 1, \ldots, m$.

Consider a finite automaton Δ having terminal alphabet V, set of states $Q = \{q_i | i = 1, \ldots, n\} \cup \{q_0, q_f\}$, and the transition function defined as $\{q_j \in \delta(q_i, x) | (\varepsilon, x, \varepsilon; j) \in R_i\}$. We assume reading x when passing from state i to state j. Clearly, Δ corresponds to the communication graph of Π.

From the construction of Δ it follows that for every $w \in L(\Delta)$ $w \in L(\Pi)$. Moreover, for every $w' \in L(\Pi)$ one may construct $w'' \in Perm(w')$ such that $w'' \in L(\Delta)$. Indeed, we may consider as $w'' \in L(\Pi)$ the string in which the insertions are performed strictly at the end of the generated string. Hence, we have $Ps(L(\Delta)) = Ps(L(\Pi))$. From the semilinearity of regular languages we have $PsSP_*(ins_1^{0,0}) \subseteq SL$.

The inverse inclusion $SL \subseteq PsLStP_*(ins_1^{0,0})$ can be shown by a direct construction of a semilinear set simulation by the graph-controlled insertion systems. First, we show that for any linear set S there is a system $\Pi' = (T, A, q_0, q_f, I, \emptyset)$, $L(\Pi) \in LStP_*(ins_1^{0,0})$ and $S = Ps(L(\Pi'))$. This system may be constructed directly from the definition of linear set. Let S be a linear set of the form

$$S = \{(x_1, \ldots, x_m) + \sum_{l=1,\ldots,s} k_l \dot{(}x_{l_1}, \ldots, x_{l_m}) \mid k_l \geq 0\}.$$

Denote by $f = \sum_{i=1\ldots m} x_i$ and by $f_l = \sum_{i=1\ldots m} x_{l_i}, l = 1, \ldots, s$. We define $T = \{a_1, \ldots, a_m\}$, $A = \{\varepsilon\}$. The communication graph consists of $f + \sum_{l=1,\ldots,s} f_l$ components.

Every component from the first $f-1$ components contains a singleton insertion rule $R_t = \{(\varepsilon, a_i, \varepsilon; t+1)\}$. There are exactly x_i components inserting a_i so that the generation of $Perm(a_i^{x_1} \ldots a_m^{x_m})$ is performed. (Clearly, the first f components compose a linear structure of the communication graph.) The component labeled by 1 is the initial and the component labeled by f is the final. To the component labeled by f there are s cycles attached: $R_f = \{(\varepsilon, a_i, \varepsilon; f+l) \mid l = 1, \ldots, s\}$. Every cycle $l = 1, \ldots, s$ consists of $\sum_{j=1,\ldots,m} x_{l_j}$ components that simulate the shuffled insertion of the word $a^{x_{l_1}} \ldots a^{x_{l_m}}$.

Since the overall effect of the i-th cycle is adding exactly $(x_{l_1}, \ldots, x_{l_m})$ to the corresponding Parikh vector we have that every word that is accepted by Π has its Parikh vector from S. The opposite direction is also true since for every vector from S we may easily construct the word from $L(\Pi)$ having the corresponding Parikh set.

Now consider an arbitrary semilinear set $SL = \cup_{i=1\ldots p} S_i$. For every linear set S_i we consider graph-controlled insertion system Π_i defined as above such that $PsL(\Pi_i)$ is equal to S_i.

Then since every Π_i has an empty axiom set we can consider the parallel merging of these systems. Hence we obtain the system that generates language with Parikh set equals to SL. □

We remark that the number of components for the system constructed in Theorem 3.3.8 depends on the length of axioms used in the system.

The following inclusion follows from the fact that insertion of m symbols (when the order in not specified) can be simulated by m components.

Lemma 3.3.9. *$PsLStP_*(ins_m^{0,0}) \subseteq SL$ for every $m \geq 1$.*

Proof. Let $\Pi = (V, A, i_0, i_f, R_1, \ldots, R_n)$ be a graph-controlled insertion system such that $L(\Pi) \in LStP_*(ins_m^{0,0})$, let $K = Ps(L)$. We may assume that the i-th component has the form $R_i = \{r : (\varepsilon, x, \varepsilon; j)\}, |x| \le n, i = 1, \ldots, m$. Consider a finite automaton Δ having terminal alphabet $B = Lab(\cup_{i=1,\ldots,n} R_i)$, set of states $Q = \{q_i | i = 1, \ldots, n\} \cup \{q_0, q_f\}$, and the transition function defined as $\{q_j \in \delta(q_i, r) | r : (\varepsilon, x, \varepsilon; j) \in R_i\}$. We assume reading the label of insertion rule r when passing from state i to state j. One may observe that Δ corresponds to the communication graph of Π. Consider also a morphism $h : B \to V^*$, defined as follows $h(r) = x$, for every $r : (\varepsilon, x, \varepsilon; j) \in R_i$. From the construction of Δ for every $w \in L(\Delta)$ it follows that $h(w) \in L(\Pi)$. Indeed, performing the insertions w at the rightmost positions we obtain $h(w)$. Moreover, for every $w' \in L(\Pi)$ one may construct $w'' \in Perm(w')$ such that $h^{-1}(w'') \in L(\Delta)$. Indeed, we just need to rearrange letters in w' according to the insertion rules. Hence, we have $Ps(h(L(\Delta))) = Ps(L(\Pi))$. Since every regular language is semilinear, and the image of a regular language under a morphism is also regular we have $PsSP_*(ins_n^{0,0}) \subseteq SL$. □

We conclude the Chapter with following remark:

Remark 3.3.10. *We remark, that if we consider only Parikh image of the context-free insertion languages then the context-free insertion of k symbols can be simulated by k one-symbol context-free insertions. For example, the rule* $r : (\varepsilon, a_1 a_2 \ldots a_k, \varepsilon; j) \in R_i,\ i, j \ge 0$ *can be simulated as follows:*

$$\begin{aligned} &r_1 : (\varepsilon, a_1, \varepsilon; i_1) \in R_i, \\ &r_2 : (\varepsilon, a_2, \varepsilon; i_2) \in R_{i_1}, \ldots, \\ &r_k : (\varepsilon, a_k, \varepsilon; j) \in R_{i_{k-1}}, \end{aligned}$$

where $i_1, \ldots, i_{k-1}$ *are the indexes of new components. Hence, we get* $PsSP_*(ins_n^{0,0}) \subseteq PsSP_*(ins_1^{0,0})$.

Chapter 4

Insertion-deletion systems

In this chapter we consider systems where both operations of contextual string insertion and of contextual string deletion are used (insertion-deletion, for short). Hence, in addition to the set of insertion rules the set of deletion rules is considered as a part of the system. Here, we systematically investigate the classes of insertion-deletion systems with respect to the size of contexts and inserted/deleted strings. We show several computationally completeness results as well as several classes that are not computationally complete. In most of the cases we compare the classes of languages generated by insertion-deletion systems with the classes of the Chomsky hierarchy.

In this chapter we also present the method of direct simulation used in all our proofs. We show that the computational completeness of many insertion-deletion systems can be reduced to the problem of modeling of the rules of another complete insertion-deletion system in the terms of the desired one. The list of results shown in this chapter is summarized in Tables 4.1 and 4.2.

4.1 Definitions

An *insertion-deletion system* is a construct $ID = (V, T, A, I, D)$, where V is a (working) alphabet, $T \subseteq V$ is a terminal alphabet, A is a finite language over V, and I, D are finite sets of triples of the form (u, α, v), $\alpha \neq \varepsilon$, where u and v are strings over V.

The elements of T are called *terminal* symbols. The symbols from $V - T$ are called nonterminals. Strings in $A \subset V^*$ are called *axioms*, the triples in I are *insertion rules* as for insertion systems, and those from D are *deletion rules*. An insertion rule $(u, \alpha, v) \in I$ indicates that the string α can be inserted between u and v, while a deletion rule $(u, \alpha, v) \in D$ indicates that α can be removed from the context (u, v). Both types of rules correspond to rewriting rules: $uv \to u\alpha v$ for the insertion rule $(u, \alpha, v) \in I$, and $u\alpha v \to uv$ for the deletion rule $(u, \alpha, v) \in D$. We denote by $\Rightarrow_{ins}$ the relation defined by an insertion rule (formally, $x \Rightarrow_{ins} y$ iff $x = x_1uvx_2, y = x_1u\alpha vx_2$, for some $(u, \alpha, v) \in I$ and $x_1, x_2 \in V^*$) and by $\Rightarrow_{del}$ the relation defined by a deletion rule (formally, $x \Rightarrow_{del} y$ iff $x = x_1u\alpha vx_2, y = x_1uvx_2$, for some $(u, \alpha, v) \in D$ and $x_1, x_2 \in V^*$). We refer by $\Rightarrow$ to any of the relations $\Rightarrow_{ins}, \Rightarrow_{del}$, and denote by $\Rightarrow^*$ the reflexive and transitive closure of $\Rightarrow$ (as usual, $\Rightarrow^+$ is its transitive closure).

The language generated by ID is defined by

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

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

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

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

If some of the parameters n, m, m', p, q, q' are not specified, then we write instead the symbol $*$. If one of the numbers from the couples m, m' and/or q, q' is equal to zero (while the other is not), then we say that corresponding families have a *one-sided context*. If all numbers m, m', q, and q' are equal to zero, then we call corresponding language families *context-free*.

Example 4.1.1. *Consider the system $ID \in INS_2^{2,2}DEL_1^{2,2}$ defined as follows*

$ID = (\{X, Y, Z, a, b\}, \{a, b\}, \{aZa\}, I, D)$, *where*

$$I = \quad \{1 : (a, aX, Za), \quad 2 : (XZ, Ya, a), \quad 3 : (a, b, Za)\};$$

$$D = \quad \{4 : (\varepsilon, X, ZY), \quad 5 : (aZ, Y, \varepsilon), \quad 6 : (b, Z, a)\}.$$

The system ID generates the language $L = \{a^n ba^n \mid n \geq 0\}$. Indeed, the system generates aba as folllows $aZa \Rightarrow_{ins} abZa \Rightarrow_{del} aba$. This derivation rewrites Z by b. The simulation of an insertion of a to the right and to the left of Z can be done by the following derivation:

$$aZa \quad \Rightarrow_{ins} \quad aaXZa \quad \Rightarrow_{ins} \quad aaXZYaa \quad \Rightarrow_{del} \quad aaZYaa \quad \Rightarrow_{del} \quad aaZaa.$$

By repeating the steps above all words of the form $a^k Za^k, k > 1$ can be obtained. At the end, aZa is rewritten to aba giving $a^k ba^k \in L(ID), k \geq 1$. Moreover, starting from the axiom aZa the only possible sequence of rules that can be applied is $(1, 2, 4, 5)^*(3, 6)$ (as the contexts of rules make their application almost deterministic) which means that no other words different from those of L can be generated.

We note that the language from Example 4.1.1 cannot be generated by insertion systems of any size [62].

The same language can be generated by another system.

Example 4.1.2. *Consider a system $ID_1 \in INS_4^{1,1}DEL_1^{0,1}$,*

$ID = (\{X, Y, a, b\}, \{a, b\}, \{XY\}, I, D)$, *where*

$$I = \{(X, aXYa, Y), (X, b, Y)\};$$

$$D = \{(\varepsilon, X, a), (\varepsilon, X, b), (\varepsilon, Y, \varepsilon)\}.$$

The system generates aba as follows $XY \Rightarrow_{ins} XaXYaY \Rightarrow_{ins} XaXbYaY \Rightarrow_{del}^4 aba$. This derivation rewrites axiom XY by aba. Similarly, by inserting k-times $aXYb$ we get the sentential form $(Xa)^k XY(aY)^k$. In order to remove all nonterminals X and Y, symbol b must be inserted in the middle. Hence, we obtain $L(ID) = \{a^k ba^k \mid k \geq 0\}$.

4.2 Normal form for insertion-deletion systems

We present below a normal form for insertion-deletion systems.

Lemma 4.2.1. *For any insertion-deletion system $ID = (V, T, A, I, D)$ having the size $(n, m, m'; p, q, q')$ it is possible to construct an insertion-deletion system $ID_2 = (V \cup \{X, Y\}, T, A_2, I_2, D_2 \cup D_2')$ having the same size such that $L(ID_2) = L(ID)$. Moreover, all rules from I_2 have the form (u, α, v), where $|\alpha| = n$, $|u| = m$, $|v| = m'$, all rules from D_2 have the form (u', α, v'), where $|\alpha| = p$, $|u'| = q$, $|v'| = q'$ and $D_2' = \{(\varepsilon, X, \varepsilon), (\varepsilon, Y, \varepsilon)\}$.*

Proof. Consider

$$\begin{aligned}
A_2 =&\{X^i w Y^t Y^j \mid w \in A, i = \max(m, q), j = \max(m', q'), t = \max(p - |w|, 0)\},\\
I_2 =&\{(z_1, xY^k, z_2) \mid (a, x, b) \in I, z_1 \in \{a \sqcup\!\sqcup X^*\}, z_2 \in \{b \sqcup\!\sqcup Y^*\}\\
&\text{and } |xY^k| = n, k \geq 0, |z_1| = m, |z_2| = m'\} \cup\\
&\cup \{(z_1, Y^n, z_2) \mid z_1, z_2 \in (V \cup \{X, Y\})^*, |z_1| = m, |z_2| = m'\},\\
D_2 =&\{(z_1, d, z_2) \mid (a, x, b) \in D, z_1 \in \{a \sqcup\!\sqcup X^*\}, z_2 \in \{b \sqcup\!\sqcup Y^*\}, d \in \{x \sqcup\!\sqcup Y^*\}\\
&\text{and } |d| = p, |z_1| = q, |z_2| = q'\}.
\end{aligned}$$

In fact, any rule having a left (resp. right) context of a smaller size is replaced by a group of rules, where the left (resp. right) context is a string over $V \cup \{X\}$ (resp. $V \cup \{Y\}$) of needed size. The same holds for the inserted or deleted symbol. Any axiom $w \in A$ is surrounded by X and Y ($X^i w Y^t Y^j$) in A_2. It is clear that if $w \in L(ID)$ then the word $X^i w' Y^j$, $w' \in \{w \sqcup\!\sqcup Y^*\}$ will be obtained in ID_2 using corresponding rules and starting from the corresponding axiom. Now symbols X and Y can be erased by rules from D_2' which implies that $w \in L(ID_2)$. It is clear that if rules from D_2' are used before this step, then at most same w may be obtained. Hence $L(ID) = L(ID_2)$. □

Next lemma shows that the deletion of terminal symbols may be excluded.

Lemma 4.2.2. *For any insertion-deletion system $ID = (V, T, A, I, D)$ there is a system $ID' = (V \cup V', T, A \cup A', I \cup I', D')$ such that $L(ID') = L(ID)$. Moreover, for any rule $(a, b, c) \in D'$ it holds that b does not contain letters from T.*

Proof. Indeed, we can transform system ID to an equivalent system ID' as follows.

Let $V' = \{N_t \mid t \in T\}$. Consider the coding function $f : V \to V \cup V'$ defined by $f(x) = N_x$ if $x \in T$ and $f(x) = x$ otherwise. Consider also the following extension to words (where *id* is the identity function):

$$F(a_1 \dots a_n) = \{g(a_1) \dots g(a_n) \mid g \in \{f, id\}\}$$

Now for any rule (a, b, c) in D (resp. in I) we introduce rules (a', b', c') in D' (resp. I'), where $a' \in F(a)$, $b' \in F(b)$ and $c' \in F(c)$. For any axiom $w \in V^*$ we add $F(w)$ to the axioms. Finally, we remove all rules $(a, b, c) \in D'$ having $|b|_T \neq 0$.

This construction insures that the nonterminal symbol N_t acts like an alias for the symbol $t \in T$, *i.e.* for any derivation producing $w_1 t w_2$ there is another derivation producing $w_1 N_t w_2$. Hence there is no difference between erasing t or N_t. This proves the statement of the lemma. □

Lemma 4.2.3. *For any insertion-deletion system $ID = (V, T, A, I, D)$ having the size $(n, m, m'; p, q, q')$ it is possible to construct an insertion-deletion system $ID_2 = (V_2 \cup \{X, Y\}, T, A_2, I_2, D_2 \cup D_2')$ having the same size such that $L(ID_2) = L(ID)$. All rules from I_2 have the form (u, α, v), where $|\alpha| = n$, $|u| = m$, $|v| = m'$, all rules from D_2 have the form (u', α', v'), where $|\alpha'| = p$, $|\alpha'|_T = 0$, $|u'| = q$, $|v'| = q'$ and $D_2' = \{(\varepsilon, X, \varepsilon), (\varepsilon, Y, \varepsilon)\}$.*

Proof. The assertion of the lemma follows from Lemmas 4.2.1 and 4.2.2. □

Definition 4.2.4. We say that an insertion-deletion system $ID = (V, T, A, I, D)$ is in a *normal form*, if it has the properties of the system ID_2 from Lemma 4.2.3.

4.3 Basic methods for computational completeness

Insertion-deletion systems represent a powerful model of computation. If the size of the system is not bounded, then an arbitrary grammar can be simulated [36].

Theorem 4.3.1. *For any type-0 grammar $G = (N, T, S, P)$ there is an insertion-deletion system $ID = (V, T, A, I, D)$ such that $L(G) = L(ID)$.*

Proof. Let $V = N \cup \{\#_i : 1 \leq i \leq Card(P)\} \cup \{\$\}$. Let $k_1 = \max\{|u|, u \to v \in P\}$ and $k_2 = \max\{|v|, u \to v \in P\}$. Consider $k = \max(k_1, k_2)$. The set A is defined as $A = \{\$^k S \$^k\}$. For any rule $i : u \to v \in P$ we add insertion rules $(xu, \#_i v, y)$, $x, y \in (N \cup \{\$\})^*$, $|xu| = k$, $|y| = k$, to I and a deletion rule $(x, u\#_i, v)$, $x \in N \cup \{\$\}$ to D. Finally, a rule $(\varepsilon, \$, \varepsilon)$ is added to D.

It is not difficult to see that such system simulates G. Indeed, for any derivation $w_1uw_2 \Rightarrow w_1vw_2$ in G there is a two-step derivation $\$^k w_1uw_2\$^k \Rightarrow \$^k w_1u\#_i vw_2\$^k \Rightarrow \$^k w_1vw_2\$^k$ in ID that simulates the corresponding production of G. If $w \in L(G)$ then the string $\$^k w\$^k$ will be obtained in ID. Additional symbols \$ can be deleted at this moment. So $w \in L(ID)$.

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

As one can see from the previous theorem, the basic idea of grammar simulation by insertion-deletion systems is a construction of a set of related insertion and deletion rules that shall be used in some specified sequence, thus performing a grammar rule simulation. Usually, insertion rules introduce new nonterminal symbols in the string which can be deleted only by the corresponding deletion rules (like the symbols $\#_i$ in theorem above). If the correct sequence is not performed, then some nonterminal symbols that cannot be deleted will remain in the string. In the subsequent sections different variants of this method are shown, thereby decreasing the size of the insertion and deletion rules.

A simulation of type-0 grammars by insertion-deletion systems is the main method to prove the computational completeness of insertion-deletion systems.

However, when several such results are established, it is much easier to prove the computational completeness by simulating other insertion-deletion systems. We call such simulation a *direct simulation.* The following theorem shows how a computationally complete insertion-deletion system can be simulated by another one. This result was firstly presented in [62], where a grammar in the Geffert normal form was simulated. Due to some errors in the proof given in the monograph we show the complete proof of the result.

The theorem uses the following method of simulation: the working (insertion or deletion) site is delimited by special symbols in order to avoid interactions between several such sites. Inside the site the sequence of insertions and deletions permits to simulate exactly one application of the corresponding rule. All additional symbols are related in such a way that the whole sequence of insertions and deletions shall be performed in order to eliminate all of them.

Theorem 4.3.2. *$INS_1^{1,1}DEL_2^{0,0} = RE$.*

Proof. We prove the theorem by simulating an insertion-deletion systems of size $(1, 1, 1; 1, 1, 1)$. It is known that these systems are computationally complete, see [65, 66]. Let $\Pi = (V, T, A, I, D)$ be an insertion-deletion system of size $(1, 1, 1; 1, 1, 1)$ in normal form. We construct a new system $\Pi' = (V_2, T, A, I_2, D_2)$ of size $(1, 1, 1; 2, 0, 0)$ such that $L(\Pi) = L(\Pi')$. In order to do this, it is enough to show that for every derivation in Π there is an equivalent derivation in Π' and conversely, so the systems generate the same terminal strings. Hence, it is enough to show how a deletion rule $(a, b, c) \in D$ from Π can be simulated using insertion and deletion rules of size $(1, 1, 1; 2, 0, 0)$. Assume that all rules in D are ordered, $n = Card(D)$, and i denotes the label of corresponding deletion rule $i : (a, b, c)$. The alphabet of Π' is defined as follows: $V_2 = V \cup \{R_i^1, R_i^2, L_i^1, L_i^2, D_i \mid i = 1 \dots n\}$.

For every deletion rule $i : (a, b, c)$ the following insertion rules are added to I_2:

$$(a, R_i^2, b) \qquad (b, R_i^1, c) \qquad (a, L_i^1, R_i^2)$$
$$(L_i^1, L_i^2, R_i^2) \qquad (L_i^1, D_i, L_i^2)$$

and the following deletion rules are added to D_2:

$$(\varepsilon, L_i^1R_i^1, \varepsilon) \qquad (\varepsilon, L_i^2R_i^2, \varepsilon), \qquad (\varepsilon, D_ib, \varepsilon).$$

We call these rules i-related. Finally, every insertion rule from I is added to I_2.

The deletion rule $i : (a, b, c)$ is simulated as follows. Firstly, $R_i^1, R_i^2, L_i^1,\ L_i^2, D_i$ are inserted:

$$w_1abcw_2 \Rightarrow w_1aR_i^2bcw_2 \Rightarrow w_1aL_i^1bR_i^1cw_2 \Rightarrow w_1aL_i^1R_i^2bR_i^1cw_2 \Rightarrow \\ w_1aL_i^1L_i^2R_i^2bR_i^1cw_2 \Rightarrow w_1aL_i^1D_iL_i^2R_i^2bR_i^1cw_2$$

Then the pairs $L_i^2R_i^2, D_ib$, and $L_i^1R_i^1$ are removed.

$$w_1aL_i^1D_iL_i^2R_i^2bR_i^1cw_2 \quad\Rightarrow\quad w_1aL_i^1D_ibR_i^1cw_2 \quad\Rightarrow\quad w_1aL_i^1R_i^1cw_2 \quad\Rightarrow\quad w_1acw_2$$

Hence every derivation in the system Π can be simulated in Π' we have $L(\Pi) \subseteq L(\Pi')$.

In order to prove that $L(\Pi') \subseteq L(\Pi)$ consider a derivation in the system Π such that an i-related rule is applied. The only rule that can modify a symbol from V is the deletion rule $(\varepsilon, D_ib, \varepsilon)$. Assume there is a pair of symbols D_ib in a sentential form wD_ibw'. We stress the point that each nonterminal from the group of i-related rules can be only deleted by the rules corresponding to the group. Let us consider possible derivations preceding the deletion of $D_ib : w_0 \Rightarrow^* wD_ibw' \Rightarrow_{del} ww'$, where w_0 is an axiom. As the only rule that inserts D_i is (L_i^1, D_i, R_i^2), we conclude that at some early point of the derivation we have the corresponding contexts

$$w_0 \Rightarrow^* w_1L_i^1D_iR_i^2bw_1' \Rightarrow^* wD_ibw'.$$

As R_i^2 can be removed only in the pair $L_2^iR_2^i$, L_1^i have to be inserted at some point, before the deletion of D_ib . In order to insert L_1^i we had to apply the rule (a, L_i^1, R_i^2) which insure that a is present to the left. Also, we may assume that the context R_i^2 is the same symbol used in the insertion of D_i. (Otherwise, we will have some symbols from V that split L_i^1 and R_i^2 and which cannot be removed with these contexts.) Finally, L_i^1 can be removed only if there is R_i^1 to the right. Since R_i^1 can be only inserted with the contexts b and c we have that once R_i^1 is inserted it can be used(in the deletion) only after every symbol to the left before L_i^1 (including b) are deleted. This implies that b and every interior symbol z, if any, are deleted as follows:

$$w_0 \Rightarrow^* w_2aL_i^1D_ibzbR_i^1cw_2' \Rightarrow^* wL_i^1D_ibR_i^1w' \Rightarrow wL_i^1R_i^1w'.$$

We remark that the deletion of each symbol in V requires at least one symbol contexts from V. Hence, L_i^1 and R_i^1 meet only if they are separated by b. Since only one D_i can be inserted, only one b can be removed. Hence, we have that every symbol from $V_2\backslash V$ is removed when all the above steps are performed. This implies that one symbol b can be removed by the i-related rules if and only if there are corresponding contexts to the right and to the left. If all the above steps are not performed, then some of additional symbols will remain in the string, hence it will never become terminal. So, we get that for every terminal derivation in Π' there is a terminal derivation in Π producing the same terminal word. Hence, $L(\Pi') = L(\Pi)$. $\square$

We give below another illustration of the method of direct simulation that we apply for a system whose computational power was not known.

Theorem 4.3.3. $INS_2^{0,0}DEL_1^{1,1} = RE$.

Proof. The proof of the theorem is based on a simulation of insertion-deletion systems of size $(2, 0, 0; 3, 0, 0)$. It is known that these systems generate any recursively enumerable language [49]. Consider $ID = (V, T, A, I, D)$ to be such a system. Now we construct a system $ID_2 = (V_2, T, A, I_2, D_2)$ of size $(2, 0, 0; 1, 1, 1)$ that will generate same language as ID.

It is clear that in order to show the inclusion $L(ID) \subseteq L(ID_2)$ it is sufficient to show how a deletion rule $(\varepsilon, abc, \varepsilon) \in D$, with $a, b, c \in V$, may be simulated by using rules of system ID_2, *i.e.*, insertion rules of type $(\varepsilon, xy, \varepsilon)$ and deletion rules of type (a', y', b'), with $a', b' \in V_2 \cup \{\varepsilon\}$, $x, y', y \in V_2$.

We may suppose that any deletion rule $(\varepsilon, abc, \varepsilon) \in D$ satisfies $a \neq b \neq c$. Indeed, if this condition does not hold, *i.e.*, we have a rule (a, a, c), then we replace this rule by an insertion rule $(\varepsilon, AA', \varepsilon)$ and two deletion rules $(\varepsilon, aA, \varepsilon)$ and $(\varepsilon, A'ac, \varepsilon)$. If a deletion rule $(\varepsilon, aaa, \varepsilon)$ is present, then it can be replaced by two insertion rules $(\varepsilon, AA', \varepsilon)$, $(\varepsilon, BB', \varepsilon)$ and three deletion rules $(\varepsilon, aA, \varepsilon)$, $(\varepsilon, A'aB, \varepsilon)$ and $(\varepsilon, B'a, \varepsilon)$.

Consider $V_2 = V \cup \{L_i, L'_i, R_i, R'_i, K_i, K'_i \mid 1 \leq i \leq Card(D)\}$.

Let us label all rules from D by integer numbers. Consider now a rule i : $(\varepsilon, abc, \varepsilon) \in D$, where $1 \leq i \leq Card(D)$ is the label of the rule. We introduce following insertion rules in I_2:

$$1 : (\varepsilon, L_iL'_i, \varepsilon) \qquad 2 : (\varepsilon, R'_iR_i, \varepsilon) \qquad 3 : (\varepsilon, K_iK'_i, \varepsilon)$$

and following deletion rules in D_2 $(l, m \in V)$:

$$\begin{array}{lll} 4 : (L_i, L'_i, a) & 5 : (L_i, a, b) & 6 : (c, R'_i, R_i) \\ 7 : (b, c, R_i) & 8 : (L_i, b, R_i) & 9 : (K_i, K'_i, L_i) \\ 10 : (K_i, L_i, R_i) & 11 : (K_i, R_i, m) & 12 : (l, K_i, m). \end{array}$$

We say that these rules are i-related.

The rule $i : (\varepsilon, abc, \varepsilon) \in D$ is simulated as follows. We first perform two insertions:

$$w_1abcw_2 \Rightarrow^1 w_1L_iL'_iabcw_2 \Rightarrow^2 w_1L_iL'_iabcR'_iR_iw_2$$

Then the following deletions

$$w_1L_iL'_iabcR'_iR_iw_2 \Rightarrow^4 w_1L_iabcR'_iR_iw_2 \Rightarrow^6 w_1L_iabcR_iw_2 \Rightarrow^5$$
$$w_1L_ibcR_iw_2 \Rightarrow^7 w_1L_ibR_iw_2 \Rightarrow^8 w_1L_iR_iw_2$$

Now we delete symbols L_iR_i using same technique as above with the help of $K_iK'_i$

$$w_1L_iR_iw_2 \Rightarrow^3 w_1K_iK'_iL_iR_iw_2 \Rightarrow^9 w_1K_iL_iR_iw_2 \Rightarrow^{10} w_1K_iR_iw_2 \Rightarrow^{11}$$
$$w_1K_iw_2 \Rightarrow^{12} w_1w_2$$

Hence, $L(ID) \subseteq L(ID_2)$. Now in order to prove the converse inclusion, we observe that we perform insertions of nonterminal symbols from V_2. After performing any of these insertions, the whole sequence of insertions and deletions above must be performed, otherwise some nonterminal symbols are left and cannot be deleted anymore.

Indeed, assume there is a derivation in which i-related rules result to another deletion sequence. Firstly, we consider the case when some symbols are inserted inside of a pair $L_iL'_i$, R'_iR_i, or $K_iK'_i$. So, L_i, R_i and K_i can be used as the contexts of the deletion rules $(5), (7), (8)$, or (11). We affirm that this will not produce any new terminal derivations. This affirmation is based on the following assertion.

Assertion 4.3.4. Assume a sentential form is produced, such that, it contains a nonterminal $X \in \{L', R', K'\}$ and X has no adjacent symbol from $\{L, R, K\}$ corresponding to the insertion rules (1),(2), and (3). Then in any following derivation, X will never be removed.

Formally, $L(ID'_2) = \emptyset$, where $ID'_2 = (V_2, T, \{w_0\}, I_2, D_2)$, and $w_0 = wdXew'$, $w, w' \in V_2^*, d, e \in V, X \in V_2 \backslash V$.

Proof. Consider the case for $X = L'_i$, for some $i > 0$. We have the sentential form $wdL'_iew', w, w' \in V_2^*, d, e \in V$. In order to remove L'_i, we must apply the rule $4 : (L_i, L'_i, a)$. Since d cannot be erased with the right context L'_i, we need to apply the insertion rule $(\varepsilon, L_iL'_i, \varepsilon)$ adjacent left to L'_i. So we have $wdL'_iew' \Rightarrow wdL_iL'_iL'_iew'$. Next, in order to remove the first L'_i the letter a must be inserted (we assume the worst case: i.e., there is an insertion rule $(\varepsilon, a, \varepsilon)$ in the system):

$$wdL_iL'_iL'_iew' \Rightarrow_{ins} wdL_iL'_iaL'_iew' \Rightarrow_{del} wdL_iaL'_iew'.$$

Next, in order to remove a we have to use a deletion rule $(\alpha, a, \beta), \alpha, \beta \in V_2$. We have two possibilities: either we remove a by the rule (L_i, a, b) from the i-related group of rules, or a new symbol is inserted such that it in turn is used as a context to remove a. In the later case the new inserted symbol that follows L_i in turn have to be removed by the similar construction. This gives an infinite repetition path. Hence a should be removed by the rule (L_i, a, b), and, hence, we must have an insertion of b before L'_i.

$$wdL_iaL'_iew' \Rightarrow_{ins} wdL_iabL'_iew' \Rightarrow_{ins} wdL_ibL'_iew'.$$

Now, it is possible to remove b in two cases, either by the rule (L_i, b, R_i), which implies the insertion of R_i by $(\varepsilon, R'_iR_i, \varepsilon)$, or by insertion of another symbols between L_i and b. The later case gives an infinite repetition of the insertion and deletion between L_i and b (as in the discussion above). Hence, we have the derivation $wdLbL'_iew' \Rightarrow_{ins} wdL_ibR'_iR_iL'_iew'$. This implies that there exists an insertion of letter $c : (\varepsilon, c, \varepsilon)$. So, we get the derivation $wdL_ibR'_iR_iL'_iew' \Rightarrow_{ins} wdL_ibcR'_iR_iL'_iew' \Rightarrow^3_{del} wdL_iR_iL'_iew'$. Now L_i and L'_i are separated by R_i. The nonterminal R_i can be removed only if K_i appears to the left of R_i, which requires another pair of symbols $L_iL'_i$ to be inserted in order to remove K'_i.

$$wdL_iR_iL'_iew' \Rightarrow wdL_iK_iK'_iL_iL'_iR_iL'_iekw' \Rightarrow wdL_iK_iL_iL'_iR_iL'_iew'.$$

In this form there are two copies of L'_i, so the first one can be removed by (only) the construction presented by the insertion of abc, whereas the second L' remains unchanged. Next, we remark that in order to remove R_i the symbol L_i must be removed firstly by the rule (K_i, L_i, R_i). But now, in order to remove the remaining K_i it must be surrounded by some letters from from V : $wdL_iK_iL_iL'_iR_iL'_iew' \Rightarrow^* wdL_id'K_ie'L'_iew'$ Clearly, this gives a repetition of the sequence of the insertions and deletions since in this form L'_i is surrounded again by symbols from V. Hence there is no way to place adjacent left to L'_i the only nonterminal used for its deletion, the nonterminal L'_i that cannot be further removed.

The cases with the nonterminals $X = R'_i$, or $X = K'_i$ can be considered similarly.

□

From the assertion we deduce that if $L_iL'_i$ is inserted then no other insertions between L_i and L'_i are possible. These nonterminals have to be removed at the end of the terminal derivation(or, at some early point). Clearly, the removal is possible by rules form the group of i-related rules. Similarly as in the assertion, we conclude that abc must follow $L_iL'_i$, and R'_iR_i is necessary inserted adjacent to the right.

Finally, in order to remove L_iR_i, $K_iK'_i$ must be inserted adjacent left to it. We remark, that at the end of the derivation K_i is removed by the rule with the contexts from $V_2\backslash V$. This implies that no other rules may be applied until the removal of abc is finished. □

4.4 One-sided insertion-deletion systems

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

Theorem 4.4.1. *$INS_1^{1,0}DEL_1^{1,2} = RE$.*

Proof. The proof of the theorem is based on a simulation of insertion-deletion systems of size $(1,1,1;1,1,1)$. It is known that these systems generate any recursively enumerable language [65]. Consider $ID = (V,T,A,I,D)$ to be such a system in normal form. Now we construct a system $ID_2 = (V_2,T,A,I_2,D_2)$ of size $(1,1,0;1,1,2)$ that will generate the same language as ID.

In order to show the inclusion $L(ID) \subseteq L(ID_2)$ it is sufficient to show how an insertion rule $(a,x,b) \in I$, with $a,b,x \in V$, may be simulated by using rules of system ID_2, *i.e.*, insertion rules of type (a',x',ε) and deletion rules of type $(a',y',b'c')$, with $a',b',c' \in V_2 \cup \{\varepsilon\}$, $x',y' \in V_2$.

We may suppose that for any rule $(a,x,b) \in I$ it holds $x \neq b$. Indeed, if this is not the case then this rule may be replaced by two insertion rules (a,B,b), (a,b,B) and one deletion rule (b,B,b).

Consider $V_2 = V \cup \{A_i, B_i \mid 1 \leq i \leq Card(I)\}$.

Let us label all rules from I by integer numbers. Consider now a rule $i : (a,x,b) \in I$, where $1 \leq i \leq Card(I)$ is the label of the rule. We add following insertion rules to I_2

$$(a,A_i,\varepsilon), \qquad (A_i,x,\varepsilon), \qquad (A_i,B_i,\varepsilon)$$

and following deletion rules to D_2

$$(x,B_i,b) \qquad (a,A_i,xB_i).$$

We say that these rules are i-related. The rule $i : (a,x,b) \in I$ is simulated as follows. We first perform insertions of A_i and x:

$$w_1abw_2 \Rightarrow w_1a(A_i)^+bw_2 \Rightarrow w_1a(A_iB_i^+)^+bw_2 \Rightarrow w_1a(A_i(x+B_i)^+)^+bw_2$$

After that we perform the deletions (they are applicable to the string $w_1aA_ixB_ibw_2$)

$$w_1aA_ixB_ibw_2 \Rightarrow w_1axB_ibw_2 \Rightarrow w_1axbw_2.$$

Hence, $L(ID) \subseteq L(ID_2)$. Now in order to prove the converse inclusion, we observe that we perform insertion of nonterminal symbols A_i and B_i from V_2. After performing this insertion, the only way to get rid of these symbols is to erase it with the introduced deletion rules. But this means that x is inserted between A_i and B_i, B_i is inserted adjacent left to b, A_i first inserts one B_i and after that one symbol x. To conclude the proof we remark that if more than one A_i, B_i or x are inserted, then it is impossible to eliminate the corresponding symbol. □

Theorem 4.4.2. *$INS_1^{1,0}DEL_2^{0,2} = RE$.*

Proof. The proof of the theorem is based on a simulation of insertion-deletion systems of size $(1,1,0;1,1,2)$ from Theorem 4.4.1. Let $ID = (V,T,A,I,D)$ be such a system in normal form. Now we construct a system $ID_2 = (V_2,T,A,I_2,D_2)$ of size $(1,1,0;2,0,2)$ that will generate the same language as ID.

It is clear that in order to show the inclusion $L(ID) \subseteq L(ID_2)$ it is sufficient to show how a deletion rule $(a,x,bc) \in D$, with $a,b,c,x \in V$, may be simulated by using rules of system ID_2, *i.e.*, insertion rules of type (a,x,ε) and deletion rules of type $(\varepsilon, x'y', b'c')$, with $b',c' \in V_2 \cup \{\varepsilon\}$, $x',y' \in V_2$.

We may suppose that for any rule $(a,x,bc) \in D$ it does not hold $x = b = c$. Indeed, if this is not the case then this rule may be replaced by an insertion rule (x, D_x, ε) and two deletion rules $(a,x,D_x x)$, (a, D_x, xx).

Consider $V_2 = V \cup \{A_i \mid 1 \leq i \leq Card(D)\}$.

Let us label all rules from D by integer numbers. Consider now a rule $i : (a,x,bc) \in D$, where $1 \leq i \leq Card(D)$ is the label of the rule. We introduce the insertion rule (a, A_i, ε) to I_2 and the deletion rule $(\varepsilon, A_i x, bc)$ to D_2. The rule $i : (a,x,bc) \in D$ is simulated as follows. We first perform insertions of A_i:

$$w_1 axbcw_2 \Rightarrow^+ w_1 a(A_i)^+ xbcw_2$$

Then we perform one deletion (it is applicable to the string $w_1 aA_i xbcw_2$)

$$w_1 aA_i xbcw_2 \Rightarrow w_1 abcw_2$$

Hence, $L(ID) \subseteq L(ID_2)$. Now in order to prove the converse inclusion, we observe that we perform insertion of nonterminal symbol A_i from V_2. After performing this insertion, the only way to get rid of this symbol is to erase it with the introduced deletion rule. But this means that x is deleted between a and b. To conclude the proof we remark that if more than one A_i is inserted, then it is impossible to eliminate the corresponding symbol A_i. □

Theorem 4.4.3. $INS_2^{0,0}DEL_2^{0,1} = RE$.

Proof. The proof of the theorem is based on a simulation of insertion-deletion systems of size $(2,0,0;3,0,0)$ from [49]. Let $ID = (V,T,A,I,D)$ be such a system in normal form. Now we construct a system $ID_2 = (V_2,T,A,I_2,D_2)$ of size $(2,0,0;2,0,1)$ that will generate the same language as ID.

It is clear that in order to show the inclusion $L(ID) \subseteq L(ID_2)$ it is sufficient to show how a deletion rule $(\varepsilon, abc, \varepsilon) \in D$, with $a,b,x \in V$, may be simulated by using rules of system ID_2, *i.e.*, insertion rules of type $(\varepsilon, xy, \varepsilon)$ and deletion rules of type $(\varepsilon, x'y', b')$ or (ε, x', b'), with $b' \in V_2 \cup \{\varepsilon\}$, $x', y' \in V_2$.

Consider $V_2 = V \cup \{A_i^{(j)}, B_i^{(j)}, \mid j \in \{1,2,3,4\}, 1 \le i \le Card(D)\}$.

Let us label all rules from D by integer numbers. Consider now a rule $i : (\varepsilon, abc, \varepsilon) \in D$, where $1 \le i \le Card(D)$ is the label of the rule. We add following insertion rules to I_2:

$$\begin{array}{lll} 1 : (\varepsilon, A_i^{(1)}B_i^{(1)}, \varepsilon) & 2 : (\varepsilon, A_i^{(2)}B_i^{(2)}, \varepsilon) & 3 : (\varepsilon, A_i^{(3)}B_i^{(3)}, \varepsilon) \\ 4 : (\varepsilon, A_i^{(4)}B_i^{(4)}, \varepsilon) & 5 : (\varepsilon, A_i^{(5)}B_i^{(5)}, \varepsilon) & \end{array}$$

and following deletion rules to D_2:

$$\begin{array}{lll} 6 : (\varepsilon, aA_i^{(1)}, B_i^{(1)}) & 7 : (\varepsilon, bA_i^{(2)}, B_i^{(2)}) & 8 : (\varepsilon, cA_i^{(3)}, B_i^{(3)}) \\ 9 : (\varepsilon, B_i^{(1)}B_i^{(2)}, A_i^{(5)}) & 10 : (\varepsilon, B_i^{(5)}B_i^{(3)}, A_i^{(4)}) & 11 : (\varepsilon, A_i^{(5)}A_i^{(4)}, B_i^{(4)}) \\ 12 : (\varepsilon, B_i^{(4)}, \varepsilon) & & \end{array}$$

The rule $i : (\varepsilon, abc, \varepsilon) \in D$ is simulated as follows. At first we perform insertions of $A_i^{(j)} B_i^{(j)}$, $j \in \{1, 2, 3, 4, 5\}$ using rules (1), ..., (5):

$$w_1 abc w_2 \Rightarrow^+ w_1 a A_i^{(1)} B_i^{(1)} b A_i^{(2)} B_i^{(2)} A_i^{(5)} B_i^{(5)} c A_i^{(3)} B_i^{(3)} A_i^{(4)} B_i^{(4)} w_2$$

After that deletion rules (6), ..., (8) are being applied:

$$w_1 a A_i^{(1)} B_i^{(1)} b A_i^{(2)} B_i^{(2)} A_i^{(5)} B_i^{(5)} c A_i^{(3)} B_i^{(3)} A_i^{(4)} B_i^{(4)} w_2 \Rightarrow^+$$
$$w_1 B_i^{(1)} B_i^{(2)} A_i^{(5)} B_i^{(5)} B_i^{(3)} A_i^{(4)} B_i^{(4)} w_2$$

Now the remaining introduced symbols are removed:

$$w_1 B_i^{(1)} B_i^{(2)} A_i^{(5)} B_i^{(5)} B_i^{(3)} A_i^{(4)} B_i^{(4)} w_2 \Rightarrow^{9,10}$$
$$w_1 A_i^{(5)} A_i^{(4)} B_i^{(4)} w_2 \Rightarrow^{11} w_1 B_i^{(4)} w_2 \Rightarrow^{12} w_1 w_2$$

Thus, we obtain string $w_1 w_2$, so we model rule $i : (\varepsilon, abc, \varepsilon) \in D$ correctly.

Hence, $L(ID) \subseteq L(ID_2)$. Now in order to prove the converse inclusion, we observe that we perform insertion of nonterminal symbols $A_i^{(j)} B_i^{(j)}$, $j \in \{1, 2, 3, 4, 5\}$ from V_2. After performing these insertion, the deletion rules above must be performed, otherwise some nonterminal symbol are left and cannot be deleted any more. More specific, assume ether one a or one b is deleted by corresponding rule 6 or 7. Then it follows that the other rule (7-th or 6-th) is also performed in the corresponding left or right adjacent position because the only way to remove $B_i^{(j)}$, $j \in \{1, 2\}$ is to perform rule 9. From this it follows that rule 5 inserts $A_i^{(5)} B_i^{(5)}$ adjacent right to $B_i^{(2)}$, with the sentential form $w_1 A_i^{(5)} B_i^{(5)} w_2$. Nonterminal $B_i^{(5)}$ can be removed only by the rule 10 which assume that rules 3 and 8 were performed adjacent right to $B_i^{(5)}$. Hence we have $w_1' A_i^{(5)} A_i^{(4)} B_i^{(4)} w_2'$. From there rules 11 and 12 can remove the remaining nonterminals. □

Next theorem present a result where insertion and deletion rules have asymmetries in the sizes of the left and the right contexts.

Theorem 4.4.4. $INS_1^{2,0} DEL_1^{0,2} = RE$.

Proof. The proof of the theorem is based on a simulation of insertion-deletion systems of size $(1,1,0;1,1,2)$. It is known that these systems generate any recursively enumerable language, see Theorem 4.4.1. Consider $ID = (V,T,A,I,D)$ to be such a system in normal form. Now we construct a system $ID_2 = (V_2,T,A,I_2,D_2)$ of size $(1,2,0;1,0,2)$ that will generate the same language as ID.

We may suppose that for any rule $(a,x,bc) \in D$ it holds $a \neq x$. If this is not the case then this rule may be replaced by two deletion rules (B,a,bc), (a,B,bc) and one insertion rule (a,B,ε), where B is a new nonterminal.

Let us label all rules from D by integer numbers. Consider now a rule $i:(a,x,bc) \in D$, where $1 \leq i \leq Card(D)$ is the label of the rule. We add the insertion rule to I_2

$$1:(ax,B_i,\varepsilon)$$

and the deletion rules

$$2:(\varepsilon,x,B_ib)$$
$$3:(\varepsilon,B_i,bc)$$

to D_2. The rule $i:(a,x,bC) \in D$ is simulated as follows. We first perform insertions of B_i:

$$w_1axbcw_2 \Rightarrow^+ w_1ax(B_i)^+bcw_2$$

Then, the deletion of x and B_i (applicable to $w_1axB_ibcw_2$)

$$w_1axB_ibw_2 \Rightarrow w_1aB_ibcw_2 \Rightarrow w_1abcw_2$$

Hence, $L(ID) \subseteq L(ID_2)$. Now in order to prove the converse inclusion, we consider the simulation of the rule $(a,x,bc) \in D$.

Here we insert B_i from V_2 corresponding to the deletion rule. It follows from rule 1 that the insertion of B_i is only possible if ax is at the left. Then there are two possible cases. The first case is to erase B_i immediately by the rule (ε,B_i,bc). Then the sentential form remains unchanged. The second case is to erase x by rule 2 and after that erase B_i. If more than one B_i is inserted, we will have to remove every additional symbol B_i by rule 3. Since we supposed $a \neq x$, no more than one x can be deleted. This gives that $L(ID_2) \subseteq L(ID)$ and hence we have $L(ID_2) = L(ID)$. □

As a corollary we obtain the following result.

Corollary 4.4.5. $INS_1^{0,2}DEL_1^{2,0} = RE$.

4.5 Uncompleteness results

In what follows we show that there are classes of one-sided insertion-deletion systems that are not computationally complete. We start with the following result.

Theorem 4.5.1. $REG \setminus INS_1^{1,0}DEL_1^{1,1} \neq \emptyset$.

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

We shall prove the above statement by contradiction. Suppose that there is such a system $\Gamma = (V, \{a,b\}, A, I, D)$ and $L(\Gamma) = L$. By Lemma 4.2.2 we can assume that Γ does not have rules that delete terminal symbols.

Consider a terminal derivation in Γ $w \Rightarrow^+ w_f$, where $w \in A$ and $w_f \in (ba)^+$. Now consider an arbitrary ba block of w_f ($w_f = \alpha ba\beta$, $\alpha, \beta \in (ba)^*$) and take its letter a. Since there are no terminal deletion rules in Γ this letter is either inserted by an insertion rule or it was a part of an axiom. We may omit the latter case by taking a derivation that produces a string that is long enough. Now suppose that this letter was inserted using a rule $(z, a, \varepsilon) \in I$, $z \in V$:

$$w \Rightarrow^* w_1zw_2 \Rightarrow w_1zaw_2 \Rightarrow^* \alpha ba\beta = w_f.$$

This means that:

$$w_1z \Rightarrow^* \alpha b$$
$$w_2 \Rightarrow^* \beta$$

Now we remark that symbol a might be inserted twice:

$$w \Rightarrow^* w_1 z w_2 \Rightarrow w_1 z a w_2 \Rightarrow w_1 z a a w_2.$$

Hence, we can obtain:

$$w \Rightarrow^* w_1 z a a w_2 \Rightarrow^* \alpha b a a \beta.$$

This contradiction concludes the proof. □

A counterpart of this result showing computational uncompleteness of systems (1,1,1;1,1,0) can be found in [51] In this case a context-free language $L = \{a^n b^n \mid n \geq 0\}$ cannot be generated.

In the way similar to Theorem 4.5.1 it is possible to show that the language $(ba)^+$ cannot be generated by systems of size either $(1,1,0;2,0,0)$ or $(2,0,0;1,1,0)$.

Theorem 4.5.2. *$REG \setminus INS_1^{1,0} DEL_2^{0,0} \neq \emptyset$.*

Proof. Consider the regular language $L = (ba)^+$. We claim that there is no insertion-deletion system Γ of size $(1,1,0;2,0,0)$ such that $L(\Gamma) = L$.

We shall prove the above statement by contradiction. Suppose that there is such a system $\Gamma = (V, \{a,b\}, A, I, D)$ in normal form and $L(\Gamma) = L$.

Consider a terminal derivation in Γ: $w \Rightarrow^+ w_f$, where $w \in A$ and $w_f \in (ba)^+$. Now consider an arbitrary ba block of w_f ($w_f = \alpha b a \beta$, $\alpha, \beta \in (ba)^*$) and take its letter a. Since there are no terminal deletion rules in Γ this letter is either inserted by an insertion rule or it was a part of an axiom. We may omit the latter case by taking a derivation that produces a string that is long enough. Now suppose that this letter was inserted using a rule $(z, a, \varepsilon) \in I$, $z \in V$:

$$w \Rightarrow^* w_1 z w_2 \Rightarrow w_1 z a w_2 \Rightarrow^* \alpha b a \beta = w_f. \tag{4.1}$$

This means that:

$$\begin{aligned} w_1 z a &\Rightarrow^* \alpha b a \\ a w_2 &\Rightarrow^* a \beta \end{aligned} \tag{4.2}$$

and the derivation of $w_1 z a$ does not depend on the derivation of $a w_2$ and vice-versa. Now we remark that symbol a might be inserted twice:

$$w \Rightarrow^* w_1 z w_2 \Rightarrow w_1 z a w_2 \Rightarrow w_1 z a a w_2. \tag{4.3}$$

From (4.3) and (4.2) we obtain:

$$w \Rightarrow^* w_1 z a a w_2 \Rightarrow^* \alpha b a a \beta.$$

This is a contradiction. □

Theorem 4.5.3. *$REG \setminus INS_2^{0,0} DEL_1^{1,0} \neq \emptyset$.*

Proof. The proof uses similar arguments as Theorem 4.5.2. We consider one more time the regular language $L = (ba)^+$. We claim there is no insertion-deletion system Γ of size (2,0,0;1,1,0) in normal form, such that $L(\Gamma) = L$.

We shall prove the above statement by contradiction. Suppose there is such a system $\Gamma = (V, \{a, b\}, A, I, D)$ and $L(\Gamma) = L$.

Consider a terminal derivation in Γ: $w \Rightarrow^+ w_f$, where $w \in A$ and $w_f \in (ba)^+$.

Now consider an arbitrary ba block of w_f ($w_f = \alpha b a \beta$, $\alpha, \beta \in (ba)^*$) and take its letter a. This letter is either inserted by an insertion rule or it was a part of an axiom. We may omit the latter case by taking a derivation that produces a string that is long enough. Now, suppose that this letter was inserted by a rule $(\varepsilon, Za, \varepsilon) \in I$, $Z \in V \setminus \{a, b\}$ (the case when a rule $(\varepsilon, aZ, \varepsilon) \in I$, $Z \in V \setminus \{a, b\}$ was used may be considered similarly):

$$w \Rightarrow^* w_1 w_2 \Rightarrow w_1 Z a w_2 \Rightarrow^* w_1' Z w' a w_2' \Rightarrow w_1' w' a w_2' \Rightarrow^* \alpha b a \beta = w_f.$$

This means that Z is deleted either by the last symbol of w_1' (i.e., $w_1' = w_1'' x$, and $(x, Z, \varepsilon) \in D$ for some $x \in V$) or, by deletion rule $(\varepsilon, Z, \varepsilon) \in D$.

Now, consider the derivation

$$\begin{aligned} w \Rightarrow^* w_1 Z a w_2 &\Rightarrow w_1 Z Z a a w_2 \Rightarrow^* w_1' Z Z w' a a w_2' \\ &\Rightarrow w_1' Z w' a a w_2' \Rightarrow w_1' w' a a w_2' \Rightarrow^* \alpha b a a \beta. \end{aligned}$$

Here we perform twice the insertion of Za. After the deletion of the first letter Z we delete the second Z by the same rule. Hence we get a contradiction as there is a terminal derivation $w \Rightarrow^* \alpha b a a \beta$. □

Now we shall concentrate on the generative power of the systems of size $(1,1,0;1,1,0)$. Firstly, we show that $INS_1^{1,0}DEL_1^{1,0}$ is included into $INS_2^{0,0}DEL_1^{1,0} \cap INS_1^{1,0}DEL_2^{0,0}$.

Lemma 4.5.4. $INS_1^{1,0}DEL_1^{1,0} \subseteq INS_1^{1,0}DEL_2^{0,0}$.

Proof. In order to prove the lemma it is enough to show that any deletion rule of size $(1,1,0)$ can be simulated by insertion rules of size $(1,1,0)$ and deletion rules of size $(2,0,0)$. Assume $(a,b,\varepsilon), a,b \in V$ is a deletion rule, where V is the alphabet of the system. Consider the insertion rule $(a,\overline{a},\varepsilon)$ and the deletion rule $(\varepsilon,\overline{a}b,\varepsilon)$, where $\overline{a} \notin V$. The deletion of b in the left context of a can be simulated as follows:

$$wabw' \Rightarrow_{ins} wa\overline{a}bw' \Rightarrow_{del} waw', w, w' \in V^*.$$

In order to show that no other word can be produced by these rules we mention that if b is not deleted immediately after insertion of $\overline{a}$, we can reorder the rules of the derivation so that the insertion and the deletion rules are performed consequently. Indeed, consider such a derivation:

$$wabw' \Rightarrow_{ins} wa\overline{a}bw' \Rightarrow_1 \ldots \Rightarrow_k w_1a\overline{a}bw_1' \Rightarrow_{del} w_1aw_1', w, w', w_1, w_1' \in V^*,$$

where $i = 1,\ldots,k$ are the steps performed before the deletion of b. Then, we can consider an equivalent derivation:

$$wabw' \Rightarrow_1 \ldots \Rightarrow_k w_1abw_1' \Rightarrow_{ins} w_1a\overline{a}bw_1' \Rightarrow_{del} w_1aw_1', w, w', w_1, w_1' \in V^*,$$

giving the same string.

Hence, every system of size $(1,1,0;1,1,0)$ can be simulated by a system of size $(1,1,0;2,0,0)$, so we get the assertion of the lemma. □

Lemma 4.5.5. $INS_1^{1,0}DEL_1^{1,0} \subseteq INS_2^{0,0}DEL_1^{1,0}$.

Proof. In order to prove the lemma it is enough to show that any insertion rule of size $(1,1,0)$ can be simulated by insertion rules of size $(2,0,0)$ and deletion rules of size $(1,1,0)$. Assume $(a,b,\varepsilon), a,b \in V$ is an insertion rule, where V is the alphabet of the system.

Consider the insertion rule $(\varepsilon, \bar{b}b, \varepsilon)$ and the deletion rule $(a, \bar{b}, \varepsilon)$, where $\bar{b}$ is a new symbol that does not belong to V. Clearly, by these rules the insertion of b in the left context of a can be simulated:

$$waw' \Rightarrow_{ins} wa\bar{b}bw' \Rightarrow_{del} wabw', w, w' \in V^*.$$

In order to show that no new words can be produced by these rules consider two cases:

1. Firstly $\bar{b}b$ is inserted, then letter a appears to the left of $\bar{b}$, and finally $\bar{b}$ is deleted.
2. Firstly $\bar{b}b$ is inserted to the right of a, then (maybe not immediately) $\bar{b}$ is deleted.

For the both cases we show that it is possible to reorder the rules of the corresponding derivation such that the insertion rule is followed by the deletion rule. Consider a derivation for the first case:

$$ww' \Rightarrow_{ins} w\bar{b}bw' \Rightarrow_{l_1} \ldots \Rightarrow_{l_k} w_1\bar{b}bw_1' \Rightarrow_{m_1} \ldots \Rightarrow_{m_k} w_2a\bar{b}bw_2' \Rightarrow_{del} w_2abw_2',$$

where $w, w_1, w_2, w', w_1', w_2' \in V^*$.

Consider a partition of rules $l_1, \ldots, l_k$ of the rules that are performed to the left of $\bar{b}$ denoted by $l_{i_1}, \ldots, l_{i_k}$, and to the right of $\bar{b}$ denoted by $l_{j_1}, \ldots, l_{j_k}$. (In the sequences we respect the order of rules.) Similarly, consider the partition of rules $m_1, \ldots, m_k$ of the rules performed to the left and to the right of $\bar{b}$, denoted by $m_{i_1}, \ldots, m_{i_k}$ and $m_{j_1}, \ldots, m_{j_k}$. We construct the following derivation:

$$ww' \Rightarrow_{l_{i_1}} \ldots \Rightarrow_{l_{i_k}} \Rightarrow_{m_{i_1}} \ldots \Rightarrow_{m_{i_k}} w_2aw' \Rightarrow_{ins} w_2a\bar{b}bw' \Rightarrow_{del}$$
$$w_2abw' \Rightarrow_{l_{j_1}} \ldots \Rightarrow_{l_{j_k}} \Rightarrow_{m_{j_1}} \ldots \Rightarrow_{m_{j_k}} w_2abw_2'.$$

Clearly, this reordering is possible since the rules of the left of $\bar{b}$ cannot affect symbols at the right of $\bar{b}$, and the rules at the right of $\bar{b}$ cannot affect symbols at the left of $\bar{b}$. Moreover, in such a derivation the insertion rule $(\varepsilon, \bar{b}b, \varepsilon)$ and the deletion rule $(a, \bar{b}, \varepsilon)$ are performed together. Hence the derivation does not produce any new word.

Now consider a derivation for the second case:

$$waw' \Rightarrow_{ins} wa\bar{b}bw' \Rightarrow_1 \ldots \Rightarrow_k w_1a\bar{b}bw'_1 \Rightarrow_{del} w_1abw'_1, w, w', w_1, w'_1 \in V^*.$$

It is clear, that once $\bar{b}$ being inserted, it can be deleted only in the left context a. Hence, we can perform the deletion of $\bar{b}$ immediately after its insertion.

Hence, every system of size $(1,1,0;1,1,0)$ can be simulated by the system of size $(2,0,0;1,1,0)$. □

We remark, that in the lemma above we order the computation in the similar way as for the context-free insertion-deletion systems in [49].

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

We start our investigations by systems that do not contain deletion rules. In the book [62] it is already shown that the family $INS_n^{1,1}DEL_0^{0,0}$, $n \geq 1$, is a subset of the family of context-free languages.

Next theorem shows that even a smaller subclass, $INS_1^{1,0}DEL_0^{0,0}$, having one-sided insertion rules contains non-regular context-free languages.

Theorem 4.5.6. $INS_1^{1,0}DEL_0^{0,0} \cap (CF \setminus REG) \neq \emptyset$.

Proof. The statement follows from Example 3.2.1. □

The below lemma shows that in the case of the family $INS_1^{1,0}DEL_0^{0,0}$ corresponding context-free grammar has a very special form.

Lemma 4.5.7. *For any $ID \in INS_1^{1,0}DEL_0^{0,0}$ it is possible to construct a context-free grammar $G = (\{S\} \cup \{S_a \mid a \in T\} \cup T, T, S, P)$, generating same language as ID and having rules of the following form:*

$$\begin{array}{ll} S \to w & w \in (\{aS_a \mid a \in T\})^* \\ S_a \to S_abS_bS_a \mid \varepsilon & a, b \in T. \end{array}$$

Proof. Consider any $ID = (T, T, A, I, \emptyset) \in INS_1^{1,0}DEL_0^{0,0}$. We construct the grammar $G = (V, T, S, P)$ as follows:

The alphabet $V = T \cup \{S_x \mid x \in T\}$. The set of productions is defined as follows.

For any $(a, b, \varepsilon) \in I$ we add following productions to P:

$$S_a \to S_a b S_b S_a \mid \varepsilon. \tag{4.4}$$

For any $a_1 \dots a_n \in A$ we add following productions to P:

$$S \to a_1 S_{a_1} \dots a_n S_{a_n} \ . \tag{4.5}$$

It is easy to observe that $L(G) = L(ID)$. Indeed, after each letter a the grammar G inserts the symbol S_a which may insert (in any order and in any combination) all possible symbols coming after the letter a. Symbol S_a corresponds to a placeholder, indicating that at that place can be a letter inserted by a. $\square$

We remark that it is possible to extend the previous lemma to systems inserting a regular language instead of a symbol.

Now we will describe the family $INS_1^{1,0}DEL_1^{1,0}$. The starting point is the construction given in Lemma 4.5.7, however it is important to show that all possible deletions may be precomputed. We start with the following definitions.

Definition 4.5.8. For any set of insertion rules I and for a letter a we define $I_a = \{x \mid (a, x, \varepsilon) \in I\}$, i.e. the set of all letters that can be inserted next to a.

Definition 4.5.9. We say that a word w is generated by the letter a if there exists a derivation $a \Rightarrow^* aw$.

Definition 4.5.10. For an insertion-deletion system $ID = (V, T, A, I, D)$ we denote by $L_{ID}(a)$ the language generated by the system $ID_a = (V, T, \{a\}, I, D)$.

We remark that any word in an insertion-deletion system of size $(1, 1, 0; 0, 0, 0)$ will have a particular structure: for any word $w = w'aw''$ of $L(ID)$ the set of words $\{w'a(L_{ID}(b))^* w''\}$, $b \in I_a$ will be also part of $L(ID)$. Hence, a letter a will be followed by a repetition of blocks $L_{ID}(b)$, $b \in I_a$.

The next lemma shows that in order to compute the effect of deletion rules for a system of size $(1, 1, 0; 1, 1, 0)$ it is enough to take only blocks containing non-repeating letters, thus giving a limit on the width of a derivation tree that should be examined in order to compute the effect of deletion rules.

Lemma 4.5.11. *Consider an insertion-deletion system $ID = (V, T, A, I, D)$ having the size $(1, 1, 0; 1, 1, 0)$. Take a letter $a \in V$ and a letter $b \in I_a$. Consider a derivation of a word w in ID:*

$$w' \Rightarrow^* z'az'' \Rightarrow^* z'\ a\ u_1b\ u_2b\ \dots\ u_nb\ u_{n+1}\ z'' \Rightarrow^* \\ \Rightarrow^* z'\ a\ u_1by_1\ u_2by_2\ \dots\ u_nby_n\ u_{n+1}\ z'' \Rightarrow^* w,$$

where $w' \in A$, $z', z'', w, y_j, u_j \in V^*$, $|u_j|_b = 0$, $1 \le j \le n+1$, and u_j is generated by a and y_j is generated by b.

If during the derivation the symbol b from the block u_iby_i, $i \ge 2$ is deleted by some symbol $d \in V$, belonging to the word $u_1by_1 \dots u_i$ then we may suppose that this d belongs to u_i ($u_i = u_i'du_i''$), i.e. b can be deleted only by a symbol from the same block.

Proof. We will show that for any derivation that does not fulfill the above property it is possible to construct another derivation which will satisfy the conditions above.

Let d not be a part of u_i. Then there are several possible cases for the position of d:

1. d belongs to by_{i-1},
2. d belongs to u_{i-1},
3. d belongs to u_kby_k, $k < i-1$,

Consider the first case. Let $by_{i-1} = xdx'$. This implies that $b \Rightarrow^* xd$, $dx' \Rightarrow^* d$ and $du_ib \Rightarrow^* d$. Then we can rearrange the derivation as follows (below we denote by v the word $u_{i+1}by_{i+1}\ \dots\ u_nby_n\ u_{n+1}$ and we underline the inserted part):

$$z'\ a\ z'' \Rightarrow^* z'\ a\ \underline{v}\ z'' \Rightarrow z'\ a\ \underline{b}\ v\ z'' \Rightarrow^* z'\ a\ b\ \underline{y_i}\ v\ z'' \Rightarrow^* z'\ a\ b\underline{xd}\ y_i\ v\ z'' \Rightarrow^* \\ \Rightarrow^* z'\ a\ \underline{u_{i-1}}bxd\ y_i\ v\ z'' \Rightarrow^* z'\ \underline{u_1by_1\ \dots u_{i-2}by_{i-2}}u_{i-1}bxd\ y_i\ v\ z''$$

The derivation above shows that it is possible to generate directly $bxdy_i$ without generating $x'u_ib$.

Now consider the second case. Let $u_{i-1} = xdx'$. Then $a \Rightarrow^* xd$, $dx'by_{i-1} \Rightarrow^* d$ and $du_i \Rightarrow^* d$. Then we can rearrange the derivation as follows (below we denote by v the word $u_{i+1}by_{i+1}\ \dots\ u_n by_n\ u_{n+1}$ and we underline the inserted part):

$$z'\ a\ z'' \Rightarrow^* z'\ a\ \underline{v}\ z'' \Rightarrow z'\ a\ \underline{b}\ v\ z'' \Rightarrow^* z'\ a\ b\underline{y_i}\ v\ z'' \Rightarrow^*$$
$$\Rightarrow^* z'\ a\ \underline{xd}\ by_i\ v\ z'' \Rightarrow z'\ a\ xd\ y_i\ v\ z'' \Rightarrow^* z'\ a\ \underline{u_1by_1\ \dots u_{i-2}by_{i-2}}\ xd\ y_i\ v\ z''$$

The above derivation satisfies the condition of the lemma, because d belongs to the same block as b.

The third case can be reduced to the second one by observing that in this case we do not need to generate the subsequence $u_{k+1}by_{k+1}\dots u_{i-1}by_{i-1}$ from a, because it is erased anyway. □

Remark 4.5.12. *We remark that the in the case of the first block u_1by_1, symbol b may be deleted by a symbol d from u_1; in this case we can extend the Lemma 4.5.11 to $i = 1$. However, d can be also from $z'a$ and this case is investigated in Lemma 4.5.15.*

Definition 4.5.13. For a word $w \in L(ID)$ ($u \Rightarrow^* w$, $u \in A$) we construct the derivation tree of w iteratively as follows:

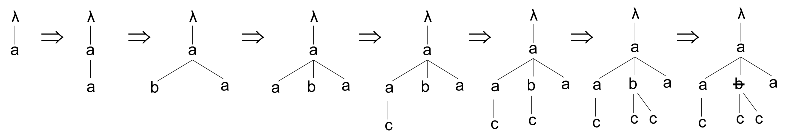
- Initially the tree has a root labeled by ε having children $a_1, \dots, a_n$, where $u = a_1 \dots a_n$. If $n = 1$, we can consider that the tree is rooted by a_1.
- For a transition $w'aw'' \Rightarrow_{ins} w'abw''$ we consider the node corresponding to the letter a above and add as a left child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

Having a derivation tree T for w, one can read w by concatenating labels of vertices from the preordering of T by a depth-first search. Hence, the root corresponds to the first letter of w and the rightmost label of the tree corresponds to the last letter of w. It is clear that there is a one-to-one correspondence between a derivation tree for an insertion-deletion system from the family $INS_1^{1,0}DEL_0^{0,0}$ and the derivation tree for the corresponding context-free grammar constructed as in Lemma 4.5.7.

Example 4.5.14. *Let $ID = (\{a,b,c\}, \{a,b,c\}, \{a\}, I, D)$ with $I = \{(a,b,\varepsilon), (a,a,\varepsilon), (b,c,\varepsilon), (a,c,\varepsilon)\}$ and $D = \{(c,b,\varepsilon)\}$. We can derive $w = aaccca$ as follows:*

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

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



The next lemma gives a bound on the depth of the derivation tree that has to be examined in order to compute the effect of deletion rules.

Lemma 4.5.15. *Consider an insertion-deletion system $ID = (V,T,A,I,D)$ having the size $(1,1,0;1,1,0)$. Consider a word $w \in L(ID)$ and the corresponding derivation tree T. Now consider that during the construction of T a deletion rule (c,x,ε) will be applied. Denote the tree at this moment T'. Denote by b the first common ancestor of deleting c and deleted x in T' and by π the path between b and the deleting c (including ends).*

If π contains multiple occurrences of c then the derivation of w may be rearranged such that the deletion of x is performed by the first occurrence of c in π.

Proof. We remark that the above situation implies that x is a child of b and π is the rightmost path in the part of the tree rooted by b and ending before x, see Figure 4.1 (a). Now if π contains several occurrences of c then we can rearrange the derivation of w as follows:

1. Use same rules until the beginning of derivation of the first element from π.
2. Derive π until the first c and the whole first subtree, see Figure 4.1(b).
3. Delete x by this c, see Figure 4.1(c).
4. Continue the derivation of π from the first c.

The obtained result is shown in Figure 4.1(d) which is exactly what should have been obtained by the application of the deletion rule (c,x,ε) on the initial tree. □

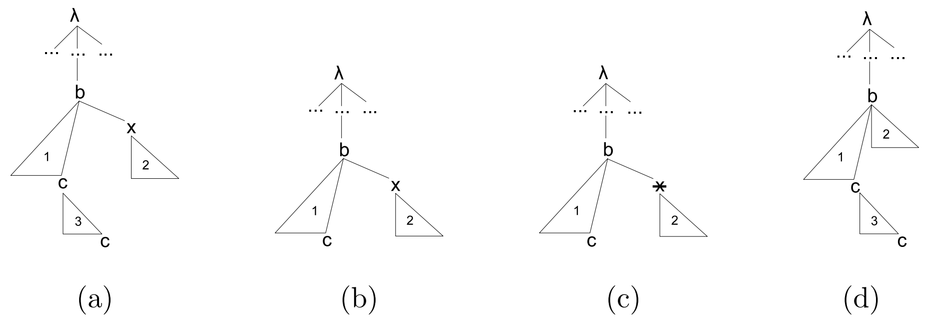


Figure 4.1: The situation from Lemma 4.5.15. The derivation tree before the application of the deletion rule (c, x, ε) (a), intermediate steps (b) and (c) and after the application (d).

Now we are ready to prove the next theorem.

Theorem 4.5.16. $INS_1^{1,0}DEL_1^{1,0} \subset CF$.

Proof. Consider an insertion-deletion system $ID = (V, T, A, I, D)$ of size $(1, 1, 0; 1, 1, 0)$. By Lemmas 4.5.11 and 4.5.15 it is possible to restrict the application of deletion rules to all possible derivation subtrees that do not have repetition of letters in width (for any node, all its children are different) and height (any path from the root does not contain repetitions of letters). Since the number of such subtrees is finite, one can precompute all possible applications of deletion rules in them.

Consider a system $ID_1 = (V, T, A, I, \emptyset)$ and construct for it a context-free grammar $G_1 = (N, T, S, P)$ as in Lemma 4.5.7. Let $G_a = (N, T, S^a, P \cup \{S^a \to aS_a\})$. Now consider any restricted (in width and height) subtree τ rooted by $a \in V \cup \{\varepsilon\}$. Let w be the word corresponding to τ. Consider the derivation tree τ' of G_a corresponding to τ and eliminate the nodes labeled by ε and edges leading to these nodes. Denote the obtained tree by τ''. Let w'' be the sentence corresponding to τ''. It is clear that w'' is a marked variant of w, the marks S_x, $x \in V$, correspond to places where $I(x)^*$ can be inserted.

Now it is possible to compute the effect of deletion rules on w'' as follows:

$$D_0^\tau = \{z \mid w'' = az\},$$
$$D_{i+1}^\tau = \Big\{uxS_tv \;\Big|\; uxzyS_tv \in D_i^\tau,\ (x,y,\varepsilon) \in D,\ t \in V,\ z \in \{S_a \mid a \in V\}^*\Big\},$$
$$D^\tau = \bigcup_{i\geq 0} D_i^\tau.$$

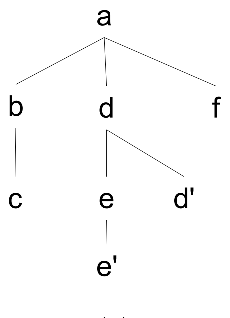
The above process is finite and D^τ contains strings corresponding all possible deletions that can be performed in τ. We define the set P_2 as follows:

$$P_2 = \{S_a \to w \mid a \in V,\ w \in D^\tau,\ \tau \text{ has the root } a\}.$$

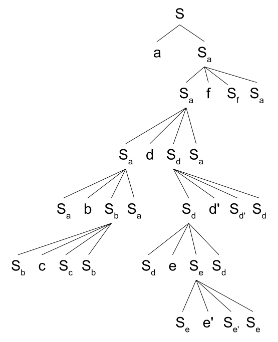
Now consider the grammar $G = (N, T, S, P \cup P_2)$. From Lemma 4.5.11 and 4.5.15 and the construction above it is clear that G simulates ID, as productions from P permit to simulate insertion rules from I and those from P_2 permit to simulate deletion rules from D. The strictness of the inclusion follows from Theorem 4.5.1, where it was shown that the language $(ba)^+$ cannot be generated by such systems.

□

Example 4.5.17. *Let $ID = (T, T, \{a\}, I, D)$ with $T = \{a, b, c, d, d', e, e', f\}$, $I = \{(a, b, \varepsilon), (a, d, \varepsilon), (a, f, \varepsilon), (b, c, \varepsilon), (d, e, \varepsilon), (d, d', \varepsilon), (e, e', \varepsilon)\}$ and $D = \{(c, d, \varepsilon), (c, e, \varepsilon)\}$. Consider the tree τ corresponding to the word $abcdee'd'f$ and the derivation tree τ'' of G_1 corresponding to τ (see Figure 4.2).*



(a)



(b)

Figure 4.2: The trees for the derivation of $abcdee'd'f$ from Example 4.5.17. The derivation in ID (a) and in G_1 (b).

Then we compute w'' and the application of rules from D to w'':

$$D_0^{\tau} = \{z \mid w'' = az\} = \{S_abS_bcS_cS_bS_adS_deS_ee'S_{e'}S_eS_dd'S_{d'}S_dS_afS_fS_a\},$$
$$D_1^{\tau} = \{S_abS_bcS_deS_ee'S_{e'}S_eS_dd'S_{d'}S_dS_afS_fS_a\},$$
$$D_2^{\tau} = \{S_abS_bcS_ee'S_{e'}S_eS_dd'S_{d'}S_dS_afS_fS_a\},$$
$$D^{\tau} = D_2^{\tau} \cup D_1^{\tau} \cup D_0^{\tau}.$$

Hence following rules shall be added to the grammar G:

$$S_a \to S_abS_bcS_deS_ee'S_{e'}S_eS_dd'S_{d'}S_dS_afS_fS_a,$$
$$S_a \to S_abS_bcS_ee'S_{e'}S_eS_dd'S_{d'}S_dS_afS_fS_a.$$

Next we present some results for systems having the contexts on the same side of insertion and deletion rules.

Theorem 4.5.18. *$REG \subset INS_1^{2,0}DEL_2^{0,0}$.*

Proof. Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton. Consider the following insertion-deletion system $ID = (V, \Sigma, A, I, D)$ of size $(1, 2, 0; 2, 0, 0)$, where

- $V = \Sigma \cup \{Q_i', Q_i'' \mid q_i \in Q\} \cup \{U\}$, where U is a new symbol,
- $A = \{Q_0''Q_0'U\}$,
- for every transition $\delta(q_i, a) = q_j$ we add to I following insertion rules:

$$(Q_i'a, Q_j', \varepsilon), \tag{4.6}$$
$$(bQ_i', a, \varepsilon), b \in \Sigma, \tag{4.7}$$
$$(Q_i''a, Q_j'', \varepsilon); \tag{4.8}$$

  for every $a \in \Sigma$, $\delta(q_0, a) = q_j$ we add to I the rule:

$$(Q_0''Q_0', a, \varepsilon), \tag{4.9}$$

- deletion rules are defined as follows:

$$D = \{(\varepsilon, Q_i', \varepsilon), (\varepsilon, Q_i'', \varepsilon) | q_i \in Q\} \tag{4.10}$$
$$\cup \ \{(\varepsilon, Q_r''U, \varepsilon) \mid q_r \in F\}. \tag{4.11}$$

We claim that $L(\mathcal{A}) = L(ID)$. First we show that $L(\mathcal{A}) \subseteq L(ID)$. Suppose $q_0 \xrightarrow{a_{i_1}} q_{i_1} \xrightarrow{a_{i_2}} \dots \xrightarrow{a_{i_r}} q_{i_r}$, $q_{i_r} \in F$ be an accepting sequence of $\mathcal{A}$. The system ID simulates the automaton $\mathcal{A}$ in two phases. First, it inserts a sequence of terminals a_i alternating with single-primed "state" nonterminals Q'_i, and after that it removes every Q'_i :

$$Q''_0 Q'_0 U \overset{(4.9),(4.6)}{\Longrightarrow} Q''_0 Q'_0 a_{i_1} Q'_{i_1} U \overset{\{(4.7),(4.6)\}^*}{\Longrightarrow} Q''_0 Q'_0 a_{i_1} Q'_{i_1} a_{i_2} \dots Q'_{i_r} U \overset{(4.10)^*}{\Longrightarrow} Q''_0 a_{i_1} a_{i_2} \dots a_{i_r} U\,.$$

In the second phase the symbols Q''_j are inserted consequently ensuring that no Q'_i has been left. Finally, every nonterminal is removed:

$$Q''_0 a_{i_1} a_{i_2} \dots a_{i_r} U \overset{(4.8)^*}{\Longrightarrow} Q''_0 a_{i_1} Q''_{i_1} a_{i_2} \dots Q''_{i_{r-1}} a_{i_r} Q''_{i_r} U \overset{(4.11)}{\Longrightarrow} Q''_0 a_{i_1} Q''_{i_1} a_{i_2} \dots Q''_{i_{r-1}} a_{i_r} \overset{(4.10)^*}{\Longrightarrow} a_{i_1} a_{i_2} \dots a_{i_r}\,.$$

By the given construction, the system ID can generate any word accepted by $\mathcal{A}$. Hence, $L(\mathcal{A}) \subseteq L(ID)$.

Now we will prove the inverse inclusion. Let $Q''_0 Q'_0 U \Rightarrow^* w = a_{i_1} a_{i_2} \dots a_{i_r}$ be a valid derivation of ID. Rules in D are defined such that the only possible way to delete the nonterminal U in the axiom is to insert a nonterminal $Q''_i \in V$ immediately at the left from U, corresponding to a final state q_i, i.e. $Q''_0 Q'_0 U \Rightarrow^* w' Q''_i U \Rightarrow^* w$, where $q_i \in F$. This can be achieved only if rules (4.8) have been applied. This ensures that the sentential form does not contain Q'_i followed by some Q''_j, for all $i, j = 1, \dots, n$. We should mention, that terminal symbols can be added into the sequential form only if it contains at least one nonterminal Q'_i.

The insertion of Q''_j is possible by rule (4.8) only if letter a can be read by the automaton after state q_i. It mimics the transition $q_i \xrightarrow{a} q_j$. Now, one can see that the sequence of rules $(4.8)^*$ can be applied to the sentential form $Q''_0 a_{i_1} a_{i_2} \dots a_{i_r} U$ starting from Q''_0 till $Q''_r U$, where $q_r \in F$. It can be easily seen that it mimics the correct sequence of transitions of $\mathcal{A}$. On the other hand, it allows to remove the nonterminal U.

Hence, $w = a_{i_1} a_{i_2} \dots a_{i_r} \in L(\mathcal{A})$. This means $L(ID) \subseteq L(\mathcal{A})$.

The strictness of the inclusion follows from Example 3.2.1 that shows there are non-regular languages in $INS_1^{1,0}DEL_0^{0,0} \subseteq INS_1^{2,0}DEL_2^{0,0}$. □

Now we consider one-sided insertion-deletion systems of the size $(1,1,0;2,3,0)$. These systems can also be compared with REG.

Theorem 4.5.19. *$REG \subset INS_1^{1,0}DEL_2^{3,0}$.*

Proof. Let $G = (N, \Sigma, S, P)$ be a regular grammar. We assume that for every production $X \longrightarrow aY \in P$, $X \in N, X \in N \cup \varepsilon, a \in \Sigma$ and $Y \neq S$.

Consider the following insertion-deletion system $ID = (V, \Sigma, A, I, D)$, where

- $V = \Sigma\ \cup \{X^i \mid X \in N, i = 1,2\} \cup \{F^1, F^2\}$, where F^1, F^2 are new symbols,
- $A = \{S^1F^2\}$,
- $I = \{(X^1, Y^1, \varepsilon), (X^1, a, \varepsilon), (X^1, X^2, \varepsilon) \mid X \longrightarrow aY \in P\}$
  $\cup\ \{(X^1, F^1, \varepsilon), (X^1, a, \varepsilon), (X^1, X^2, \varepsilon) \mid X \longrightarrow a \in P\}$,
- $D = \{(X^1X^2a, Y^1Y^2, \varepsilon) \mid X \longrightarrow aY \in P\}$
  $\cup\ \{(X^1X^2a, F^1F^2, \varepsilon) \mid X \longrightarrow a \in P\} \cup \{(\varepsilon, S^1S^2, \varepsilon)\}$.

We claim that $L(G) \subseteq L(ID)$. Indeed, consider a terminating derivation $S \Rightarrow a_1X_1 \Rightarrow a_1a_2X_2 \Rightarrow^* a_1a_2\dots a_{n-1}X_{n-1} \Rightarrow a_1a_2\dots a_n$.

The following derivation in ID simulates it. First, we use only insertions (we underline the inserted symbols):

$$S^1F^2 \Rightarrow S^1\underline{S^2a_1X_1^1}F^2 \Rightarrow S^1S^2a_1X_1^1\underline{X_1^2a_2X_2^1}F^2 \Rightarrow \dots \Rightarrow$$
$$S^1S^2a_1X_1^1\dots X_{n-2}^1\underline{X_{n-2}^2a_{n-1}X_{n-1}^1}F^2 \Rightarrow$$
$$S^1S^2a_1X_1^1\dots X_{n-1}^1\underline{X_{n-1}^2a_nF^1}F^2.$$

Then we remove the pairs of nonterminals starting from the rightmost pair F^1F^2 until the leftmost pair S^1S^2 (we underline the symbols to be deleted):

$$S^1S^2a_1X_1^1X_1^2a_2\dots X_{n-1}^2a_n\underline{F^1F^2} \Rightarrow$$
$$S^1S^2a_1X_1^1X_1^2a_2\dots a_{n-1}\underline{X_{n-1}^1X_{n-1}^2}a_n \Rightarrow \dots \Rightarrow$$
$$S^1S^2a_1\underline{X_1^1X_1^2}a_2\dots a_n \Rightarrow \underline{S^1S^2}a_1a_2\dots a_n \Rightarrow a_1a_2\dots a_n\,.$$

Now we prove that $L(ID) \subseteq L(G)$. First, we mention that in any terminating derivation $S^1F^2 \Rightarrow^* w, w \in T^*$, F^1 must appear adjacent to F_2 since the only way to remove F_2 is to use the rule $(Y^1Y^2a, F^1F^2, \varepsilon) \in D$. This rule corresponds to the production $X \to a$. Now, in order to remove Y^1Y^2 we must use the rule $(X^1X^2a, Y^1Y^2, \varepsilon)$ corresponding to the production $X \to aY$. Clearly, we perform the deletions from right to left until we delete S^1S^2. Hence, every terminal derivation by ID corresponds to some derivation in G. Thereafter, we obtain $L(ID) = L(G)$.

The strictness of the inclusion follows from Example 3.2.1. $\square$

Example 4.5.20.

$$(ab)^+ \in INS_1^{1,0}DEL_1^{2,0}$$

Consider the insertion-deletion system $ID = (N, T, A, I, D)$ *of size* $(1, 1, 0, 1, 2, 0)$ *defined as follows*

$$\begin{aligned} N =&\{A, B, F, a, b\}, \quad && T = \{a, b\}, \quad && A = \{abAF\}; \\ I =&\{1 : (A, B, \varepsilon), && 2 : (A, a, \varepsilon), && 3 : (B, A, \varepsilon), \quad && 4 : (B, b, \varepsilon)\}; \\ D =&\{5 : (ab, A, \varepsilon), && 6 : (ba, B, \varepsilon), && 7 : (ab, F, \varepsilon)\}. \end{aligned}$$

We claim that $L(ID) = (ab)^+$. *The inclusion* $(ab)^+ \subseteq L(ID)$ *can be shown by the following derivations in* ID :

$$abAF \overset{5,7}{\Rightarrow} ab;$$

and

$$abAF \overset{1,2,5}{\Rightarrow} abaBF \overset{3,4,6}{\Rightarrow} (ab)^2AF \Rightarrow \overset{(1,2,5,3,4,6)^{k-1}}{\cdots} \Rightarrow (ab)^{k+1}AF \overset{5,7}{\Rightarrow} (ab)^{k+1},$$

for any $k \geq 1$.

Now in order to show that $L(ID) \subseteq (ab)^+$ *we mention that rules 5 and 6 require alternation of the symbols* a *and* b. *Otherwise some nonterminals may be left in the string.*

4.6 Descriptional complexity

We collect in tables below our results as well as best known results on insertion-deletion systems. The first table contains the systems with both contexts of same size and the second table concentrates on complexity of systems with one-sided contexts. We do not present the symmetrical variants (by interchanging sizes of right and left contexts) which have same generation capabilities.

Table 4.1: Known results on insertion-deletion systems

Nb.	$(n, m, m'; p, q, q')$	size	family	references
1	$(2, 0, 0; 3, 0, 0)$	5	RE	[49]
2	$(3, 0, 0; 2, 0, 0)$	5	RE	[49]
3	$(1, 1, 1; 2, 0, 0)$	5	RE	[62], Theorem 4.3.2
4	$(1, 1, 1; 1, 1, 1)$	6	RE	[65, 66]
5	$(2, 0, 0; 2, 0, 0)$	4	$\subsetneq CF$	[67]
6	$(m, 0, 0; 1, 0, 0)$	m+1	$\subsetneq CF$	[67]
7	$(1, 0, 0; p, 0, 0)$	p+1	$\subsetneq REG$	[67]
8	$(2, 0, 0; 1, 1, 1)$	5	RE	Theorem 4.3.3

Table 4.2: Known results on insertion-deletion systems with one-sided contexts

Nb.	$(n, m, m'; p, q, q')$	size	family	references
9	$(1, 1, 2; 1, 1, 0)$	6	RE	[51]
10	$(2, 0, 2; 1, 1, 0)$	6	RE	[51]
11	$(2, 0, 1; 2, 0, 0)$	5	RE	[51]
12	$(1, 1, 1; 1, 1, 0)$	5	$\subsetneq RE$	[51]
13	$(1, 1, 0; 1, 1, 2)$	6	RE	Theorem 4.4.1
14	$(1, 1, 0; 2, 0, 2)$	6	RE	Theorem 4.4.2
15	$(2, 0, 0; 2, 0, 1)$	5	RE	Theorem 4.4.3
16	$(1, 1, 0; 1, 1, 1)$	5	$\subsetneq RE$	Theorem 4.5.1
17	$(1, 2, 0; 1, 0, 2)$	6	RE	Theorem 4.4.4
18	$(1, 1, 0; 1, 1, 0)$	4	$\subsetneq CF$	Theorem 4.5.16
19	$(1, 0, 2; 2, 0, 0)$	5	$\supsetneq REG$	Theorem 4.5.18
20	$(1, 1, 0; 2, 3, 0)$	7	$\supsetneq REG$	Theorem 4.5.19

Chapter 5

Graph-controlled insertion-deletion systems

In the previous chapter it was shown that there are classes of insertion-deletion systems that cannot generate RE. Making an analogy to context-free grammars, a natural extension of insertion-deletion systems using the graph-controlled approach can be done. Such model introduces states (or labels of the program) associated to every insertion or deletion rule. The transition is performed by applying corresponding rule and choosing the new state (thus the rule to be applied) among a specific set of rules. Another definition of this model in the style of [60] or [16] can be done. This definition supposes that there are disjoint groups of insertion and deletion rules (corresponding to *membranes* from [60] or *components* from [16]). The transition is performed by firstly choosing and applying one of applicable rules from the current group and switching to the next group indicated in the rule description. This definition can be easily reduced to the first one. This is why we shall consider that in the subsequent text we use the second definition. We also remark that the last definition almost coincides with the definition of insertion-deletion P systems [60]. Moreover, all our results on graph-controlled insertion-deletion systems were obtained under the name of insertion-deletion P systems. In this chapter we show that the introduced graph-controlled variant of insertion-deletion systems permits in most of the cases to increase the computational power of corresponding insertion-deletion systems.

5.1 Definitions

A *graph-controlled insertion-deletion system* $\Pi = (V, T, A, i_0, i_f, R_1, \dots, R_n)$ is a particular case of graph-controlled scheme where the set of operations is defined as a set of contextual insertion-deletion rules. Components $R_i, i = 1 \dots n$ contain rules of the form $r : (u, \alpha, v; j)_t \in R_i$, where $i, j \in 1 \dots n$ are labels of components The triplet $(u, \alpha, v) \in V^* \times V^* \times V^*$ is an insertion rule if $t = a$, and is a deletion rule if $t = e$.

The computation is performed like in the case of graph-controlled insertion system. We remark that, as in the case of insertion-deletion systems we may consider that the insertion and deletion rules are in the normal form.

We would like to note that the presentation of graph-controlled insertion-deletion systems has much in common with the definition of insertion-deletion P systems [60]. There is one-to-one correspondence between components of graph-controlled and membranes of P systems on a graph structure. The differences are the following: axioms of graph-controlled insertion-deletion systems are present only in component i_0, while the initial state of insertion-deletion P systems is determined by the contents of all membranes; the final component of P systems (in the case of membrane tree structure) has no rules and considered as the environment of the system.

Another difference is that insertion-deletion P systems are often considered as a model working in maximal parallel mode, while the graph-controlled insertion-deletion systems are rather sequential one whose configuration is determined by a single word in some component. We note that the notions of insertion-deletion P systems and graph-controlled insertion-deletion systems, can be easily transformed to each other, however, in what follows, we will use the graph-controlled variant, because of a much simpler definition, and shorter representation of proofs.

We denote by $ELSP_k(ins_p^{m,m'}, del_p^{q,q'})$ the family of languages generated by graph-controlled insertion-deletion systems with $k \geq 1$ components and insertion and deletion rules of size at most $(n, m, m'; p, q, q')$ and whose communication graph has a tree structure. We omit the letter E if $T = V$.

As in Chapter 4, the letter t is inserted before P to denote classes whose communication graph is arbitrary, e.g., $ELStP_k(ins_p^{m,m'}, del_p^{q,q'})$.

Example 5.1.1. *It is easy to see that language* $\{a^nba^n \mid n \geq 1\}$ *is generated by the following graph-controlled insertion-deletion system*

$$(\{a,b\}, \{a,b\}, \{aba\}, 1, 1, R_1, R_2) \in LSP_2(ins_1^{1,1}, del_0^{0,0}),$$

where $R_1 = \{(\varepsilon, a, b; 2)_a\}$*, and* $R_2 = \{(b, a, \varepsilon; 1)_a\}$*.*

We remark that the language generated by the system from Example 5.1.1 cannot be generated by any insertion system of size $(n, 1, 1)$, for any $n \geq 1$.

Example 5.1.2. *Consider the following example:*
$\Pi_1 = (\{a,b,c\}, \{a,b,c\}, \{\varepsilon\}, 1, 1, R_1, R_2, R_3)$*, where*

$$R_1 = \{(\varepsilon, a, \varepsilon; 2)_a\}; R_2 = \{(\varepsilon, b, \varepsilon; 3)_a\}; R_2 = \{(\varepsilon, c, \varepsilon; 1)_a\}.$$

Clearly, $\Pi_1 \in ELStP_3(ins_1^{0,0}, del_0^{0,0})$*. One can see that* Π_1 *generates the non context-free language.* $L = \{w \mid |w|_a = |w|_b = |w|_c\}$*. Indeed, every cycle that activates components* $1-2-3$ *inserts exactly one symbol* a, b *and* c*.*

For the tree case the language $\{w \in \{a,b\}^* \mid |w|_a = |w|_b\}$ can be generated in a similar manner.

5.2 The computational power of small systems

We start with the following result.

Theorem 5.2.1. $PsStP_*(ins_1^{0,0}, del_1^{0,0}) = PsMAT^\lambda$*.*

Proof. The inclusion $PsStP_*(ins_1^{0,0}, del_1^{0,0}) \subseteq PsMAT^\lambda$ follows from the simulation of minimal graph-controlled insertion-deletion without contexts systems by partially blind register machines [22]. Indeed, any rule $(\varepsilon, a, \varepsilon; q)_a \in R_p$ can be simulated by the instruction $p : (ADD(a), q)$. Similarly, any rule $(\varepsilon, a, \varepsilon; q)_e \in R_p$ can be simulated by the instruction $p : (SUB(a), q)$.

The final component i_0 is associated to the final state, while the condition that a string does not contain non-terminal symbols is represented by requirement that all register except output ones are zero.

The converse inclusion follows from the simulation of partially blind register machines by graph-controlled insertion-deletion systems. Indeed, with every instruction p of the register machine we associate a component. Instruction $p : (ADD(A_k), q)$ is simulated by rule $(\varepsilon, A_k, \varepsilon; q)_a \in R_p$, and instruction $p : (SUB(A_k), q)$ by $(\varepsilon, A_k, \varepsilon; q)_e \in R_p$. The requirement that non-output registers are empty corresponds to the condition that the resulting string does not contain non-terminal symbols. □

If the communication graph is a tree, one-way inclusion follows as a particular case. Non-extended systems are also a particular case.

Corollary 5.2.2. *$PsSP_*(ins_1^{0,0}, del_1^{0,0}) \subseteq PsMAT^\lambda$.*

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

Theorem 5.2.3. *$REG \backslash ELStP_*(ins_n^{0,0}, del_1^{0,0}) \neq \emptyset$, for any $n > 0$.*

However, as shown in Example 5.1.2, there are non-context-free languages that can be generated by such systems (even without deletion).

Now, we show a more general inclusion:

Theorem 5.2.4. *$ELStP_*(ins_n^{0,0}, del_1^{0,0}) \subset MAT^\lambda$, for any $n > 0$.*

Proof. Consider a graph-controlled insertion-deletion system $\Pi = (V, T, w, p_0, h, R)$ with rules in normal form and $H = Lab(R)$. Such a system can be simulated by the following matrix grammar $G = (V \cup H, T, S, P)$.

For every insertion rule $(\varepsilon, a_1 \cdots a_n, \varepsilon; q)_a$ in component p, let P contain the matrix $\{p \to q, D \to Da_1D \cdots Da_nD\}$. For any deletion instruction $(\varepsilon, A, \varepsilon; q)_e$ in component p, let P contain the matrix $\{p \to q, A \to \varepsilon\}$. We also add to P

three additional matrices: $\{h \to \varepsilon\}$, $\{D \to \varepsilon\}$ and $\{S \to p_0 D a_1 D \cdots D a_m D\}$ ($w = a_1 \cdots a_m$).

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

Now we show that using contexts any RE language can be generated.

Theorem 5.2.5. *$ELSP_5(ins_1^{1,0}, del_1^{1,0}) = RE$.*

Proof. Let $G = (N, T, S, P)$ be a type-0 grammar in Kuroda normal form. We show that there is a system $\Pi \in ELSP_5(ins_1^{1,0}, del_1^{1,0})$ such that $L(\Pi) = L(G)$. Suppose that rules in P are ordered and $n = Card(P)$. Consider a graph-controlled insertion-deletion system $\Pi = (V, T, \{S\}, i_0, i_f, R_1, \ldots, R_5)$, where $V = N \cup T \cup \{P_1^i, P_2^i \mid 1 \le i \le n\} \cup \{X\}$ the initial and the final components are equal $i_0 = i_f = 1$, and the communication graph has the (tree) structure presented in Figure 5.1.

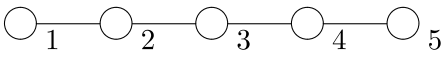


Figure 5.1: Communication graph for Theorem 5.2.5.

For any context-sensitive production $i : AB \to CD \in P$ consider the rules:

$$\begin{aligned}
R_1^i =& \{r_i.1.1 : (\varepsilon, P_1^i, \varepsilon; 2)_a\};\\
R_2^i =& \{r_i.2.1 : (P_1^i, A, \varepsilon; 3)_e, \quad && r_i.2.2 : (\varepsilon, P_2^i, \varepsilon; 1)_e\};\\
R_3^i =& \{r_i.3.1 : (P_1^i, B, \varepsilon; 4)_e, && r_i.3.2 : (P_2^i, C, \varepsilon; 2)_a\};\\
R_4^i =& \{r_i.4.1 : (P_1^i, P_2^i, \varepsilon; 5)_a, && r_i.4.2 : (P_2^i, D, \varepsilon; 3)_a\};\\
R_5^i =& \{r_i.5.1 : (\varepsilon, P_1^i, \varepsilon; 4)_e\}.
\end{aligned}$$

For any context-free production $i : A \to BC \in P$ consider the rules

$$\begin{aligned}
R_1^i =& \{r_i.1.1 : (\varepsilon, P_1^i, \varepsilon; 2)_a\};\\
R_2^i =& \{r_i.2.1 : (P_1^i, A, \varepsilon; 3)_e, \quad && r_i.2.2 : (\varepsilon, P_2^i, \varepsilon; 1)_e\};\\
R_3^i =& \{r_i.3.1 : (P_1^i, X, \varepsilon; 4)_a, && r_i.3.2 : (P_2^i, B, \varepsilon; 2)_a\};\\
R_4^i =& \{r_i.4.1 : (P_1^i, P_2^i, \varepsilon; 5)_a, && r_i.4.2 : (P_2^i, C, \varepsilon; 3)_a\};\\
R_5^i =& \{r_i.5.1 : (\varepsilon, P_1^i, \varepsilon; 4)_e\}.
\end{aligned}$$

For any production $i : A \to \alpha \in P, \alpha \in T$ consider the rules

$$\begin{aligned}
R_1^i =& \{r_i.1.1 : (\varepsilon, P_1^i, \varepsilon; 2)_a\};\\
R_2^i =& \{r_i.2.1 : (P_1^i, A, \varepsilon; 3)_e, \qquad r_i.2.2 : (\varepsilon, P_2^i, \varepsilon; 1)_e\};\\
R_3^i =& \{r_i.3.1 : (P_1^i, P_2^i, \varepsilon; 4)_a, \qquad r_i.3.2 : (P_2^i, \alpha, \varepsilon; 2)_a\};\\
R_4^i =& \{r_i.4.1 : (\varepsilon, P_1^i, \varepsilon; 3)_e\};\\
R_5^i =& \emptyset.
\end{aligned}$$

For every production $i : A \to \varepsilon \in P$ consider the rules

$$R_1^i = \{i.1.1 : (\varepsilon, A, \varepsilon; 1)_e, \}; R_2^i = R_3^i = R_4^i = R_5^i = \emptyset.$$

Now associate with $k-$th component $k = 2, \dots, 5$ the set of rules $R_k = \cup_{i=1,\dots,n} R_k^i$ and with component 1 the set of rules $R_1 = \cup_{i=1,\dots,n} R_1^i \cup \{0 : (\varepsilon, X, \varepsilon; 1)_e\}$. We claim that Π generates the same language as G.

We prove that every step of the derivation in G can be simulated in Π. Consider as example the production $i : AB \to CD \in R$. The simulation of this production is done as follows. Let a string w_1ABw_2 is in the component 1. Then, there is the following derivation

$$\begin{aligned}
&(1, w_1ABw_2) \Rrightarrow_{r_i.1.1} (2, w_1P_1^iABw_2) \Rrightarrow_{r_i.2.1} (3, w_1P_1^iBw_2) \Rrightarrow_{r_i.3.1}\\
&\quad (4, w_1P_1^iw_2) \Rrightarrow_{r_i.4.1} (5, w_1P_1^iP_2^iw_2) \Rrightarrow_{r_i.5.1} (4, w_1P_2^iw_2) \Rrightarrow_{r_i.4.2}\\
&\qquad (3, w_1P_2^iDw_2) \Rrightarrow_{r_i.3.2} (2, w_1P_2^iCDw_2) \Rrightarrow_{r_i.2.2} (1, w_1CDw_2).
\end{aligned}$$

The simulation of other productions is done in a similar way.

It is easy to see that after the insertion of P_1^i by the rule $r_i.1.1$ all rules corresponding to i-th production have to be applied until the rule $r_i.2.2$, otherwise the derivation will be blocked. We also note that every sentential form has at most one copy of symbols P_1^i and $P_2^i, i = 1, ..., n$ and none of them is present if the component 1 is active.

Productions of the form $A \to BC$, where $A, B, C \in N$ are controlled by P_i^1 that pushes the sequential form "right" from R_1 to R_5, and by P_i^2 that controls the sequence of rules pushing the sequential form "left" from R_5 to R_1. It is simulated as follows:

$$\begin{aligned}&(1, w_1Aw_2) \Rrightarrow_{r_i.1.1} (2, w_1P_1^iAw_2) \Rrightarrow_{r_i.2.1} (3, w_1P_1^iw_2) \Rrightarrow_{r_i.3.1}\\ &\quad (4, w_1P_1^iP_2^iw_2) \Rrightarrow_{r_i.4.1} (5, w_1P_1^iP_2^iXw_2) \Rrightarrow_{r_i.5.1} (4, w_1XP_2^iw_2) \Rrightarrow_{r_i.4.2}\\ &\qquad (3, w_1XP_2^iCw_2) \Rrightarrow_{r_i.3.2} (2, w_1XP_2^iBCw_2) \Rrightarrow_{r_i.2.2} (1, w_1XBCw_2).\end{aligned}$$

We mention that one extra symbol $X \notin N \cup T$ is introduced by the rule $r_i.3.1$. In fact the context-free production $A \to BC$ is simulated by two equivalent productions $A \to XBC$ and $X \to \varepsilon$ with a special nonterminal X. This is due to the fact that the total number of inserted and deleted symbols must be even. This symbol is finally deleted by the rule $0 : (\varepsilon, X, \varepsilon, 1)_e \in R_1$.

We use the same technique for productions $A \to \alpha, A \in N, \alpha \in T \cup N$:

$$\begin{aligned}&(1, w_1Aw_2) \Rrightarrow_{r_i.1.1} (2, w_1P_1^iAw_2) \Rrightarrow_{r_i.2.1}\\ &\quad (3, w_1P_1^iw_2) \Rrightarrow_{r_i.3.1} (4, w_1P_1^iP_2^iw_2) \Rrightarrow_{r_i.4.1}\\ &\qquad (3, w_1P_2^iw_2) \Rrightarrow_{r_i.3.2} (2, w_1P_2^i\alpha w_2) \Rrightarrow_{r_i.2.2} (1, w_1\alpha w_2)\end{aligned}$$

The result is collected in the final component of the system and the strings are sent there using the rule $(\varepsilon, X, \varepsilon; 1)_e$ in the first component. To finish the proof we observe that every correct sentential form has at most one symbol P_i^1 and $P_i^2, i = 1, ..., n$. After the insertion of P_i^1 in the first component either all rules corresponding to i-th rule have to be applied (in the defined order) or the derivation is blocked. Hence, we have $L(G) = L(\Pi)$. □

Symbols P_1^i can be inserted by rules with a right context. This gives a characterization of RE languages by the family $ELSP_5(ins_1^{0,1}, del_1^{1,0})$, *i.e.* with contexts for insertion and deletion on different sides. In order to achieve this, the pair of rules $r_i.2.1, r_i.3.1$ corresponding to i-th context-sensitive production is replaced by new rules $r_i.2.1 : (\varepsilon, P_1^i, B; 3)_e$, and $r_i.3.1 : (\varepsilon, P_1^i, A; 4)_e$ respectively, and for every context-free and terminal production j the rule $r_j.2.1$ is replaced by $r_j.2.1 : (\varepsilon, P_1^i, A; 3)_e$. Taking also into account the symmetrical cases we get:

Corollary 5.2.6. $ELSP_5(ins_1^{1,0}, del_1^{0,1}) = ELSP_5(ins_1^{0,1}, del_1^{1,0}) = ELSP_5(ins_1^{0,1}, del_1^{0,1}) = RE.$

Theorem 5.2.7. *$ELSP_5(ins_1^{1,0}, del_2^{0,0}) = RE$.*

Proof. For a grammar $G = (N, T, S, P)$ consider the system

$$\Pi = (V, T, \{S\}, i_0, i_f, R_1, \dots, R_5)$$

having the same communication graph as in Theorem 5.2.5. We define the working alphabet as $V = N \cup T \cup \overline{P}$, where $\overline{P} = \{P_1^i, P_2^i, P_3^i \mid 1 \le i \le n\}$ and n is the number of productions in the grammar.

For every context sensitive rule $i : AB \to CD \in P$ consider:

$$\begin{aligned}
R_1^i =& \{r_i.1.1 : (A, P_1^i, \varepsilon; 2)_a\}; \\
R_2^i =& \{r_i.2.1 : (P_1^i, P_2^i, \varepsilon; 3)_a, \qquad r_i.2.2 : (\varepsilon, P_1^iP_3^i, \varepsilon; 1)_e\}; \\
R_3^i =& \{r_i.3.1 : (\varepsilon, P_2^iB, \varepsilon; 4)_e, \qquad r_i.3.2 : (P_3^i, C, \varepsilon; 2)_a\}; \\
R_4^i =& \{r_i.4.1 : (P_1^i, P_3^i, \varepsilon; 3)_a\}; \\
R_5^i =& \emptyset.
\end{aligned}$$

For every context-free production $i : A \to BC \in P$ consider:

$$\begin{aligned}
R_1^i =& \{r_i.1.1 : (A, P_1^i, \varepsilon; 2)_a\}; \\
R_2^i =& \{r_i.2.1 : (P_1^i, P_2^i, \varepsilon; 3)_a, \qquad r_i.2.2 : (\varepsilon, P_2^i, \varepsilon; 1)_e\}; \\
R_3^i =& \{r_i.3.1 : (P_1^i, B, \varepsilon; 4)_a, \qquad r_i.3.2 : (\varepsilon, P_3^i, \varepsilon; 2)_e\}; \\
R_4^i =& \{r_i.4.1 : (\varepsilon, AP_1^i, \varepsilon; 5)_e, \qquad r_i.4.2 : (P_3^i, C, \varepsilon; 3)_a\}; \\
R_5^i =& \{r_i.5.1 : (P_2^i, P_3^i, \varepsilon; 4)_a\}.
\end{aligned}$$

For every terminal production $i : A \to \alpha \in P$ consider:

$$\begin{aligned}
R_1^i =& \{r_i.1.1 : (A, P_1^i, \varepsilon; 2)_a\}; \\
R_2^i =& \{r_i.2.1 : (P_1^i, \alpha, \varepsilon; 3)_a, \qquad r_i.2.2 : (\varepsilon, P_2^iP_3^i, \varepsilon; 1)_e\}; \\
R_3^i =& \{r_i.3.1 : (P_1^i, P_2^i, \varepsilon; 4)_a, \qquad r_i.3.2 : (P_2^i, P_3^i, \varepsilon; 2)_a\}; \\
R_4^i =& \{r_i.4.1 : (\varepsilon, AP_1^i, \varepsilon; 3)_e\}; \\
R_5^i =& \emptyset.
\end{aligned}$$

For every $\varepsilon-$production $i : A \to \varepsilon \in P$ consider

$$R_1^i = \{r_i.1.1 : (\varepsilon, A, \varepsilon; 1)_e\}; \quad R_2^i = R_3^i = R_4^i = R_5^i = \emptyset.$$

We associate with $k-$th component the set of rules $R_k = \bigcup_{i=1,\dots,n} R_k^i$. The simulation of i-th production is controlled by symbols P_1^i, P_2^i, P_3^i. Once a rule from R_1 is applied, inserting P_1^i for some $1 \le i \le n$, the only possible continuation of the derivation is to perform the sequence of rules corresponding to i-th production. We apply the same argument as in Theorem 5.2.5 in order to verify the equivalence $L(G) = L(\Pi)$, i.e., every step in derivation by grammar G can be simulated in Π_1 and that no other strings are produced.

Consider a context-sensitive production $i : AB \to AC \in P$. The simulation of this production is controlled by symbols P_i^1, P_i^2 and P_i^3. We assume that any sentential form whenever first component is active does not contain symbols from $\overline{P}$. Consider a configuration $(1, w_1ABw_2)$, where $w_1, w_2 \in V^*$ The following derivation rewrites AB by AC:

$$\begin{aligned}(1, w_1ABw_2) &\Rrightarrow_{r_i.1.1} (2, w_1AP_i^1Bw_2) \Rrightarrow_{r_i.2.1}\\ &(3, w_1AP_i^1P_i^2Bw_2) \Rrightarrow_{r_i.3.1} (4, w_1AP_i^1w_2) \Rrightarrow_{r_i.4.1} (3, w_1AP_i^1P_i^3w_2) \Rrightarrow_{r_i.3.2}\\ &\qquad (2, w_1AP_i^1P_i^3Cw_2) \Rrightarrow_{r_i.2.2} (1w_1ACw_2)\end{aligned}$$

For the sentential form $(2, w_1AP_i^1P_i^3Cw_2)$, rule $r_i.2.1$ can be also applied. This insertion leads to an infinite loop:

$$\begin{aligned}(2, w_1AP_i^1P_i^3Cw_2) &\Rrightarrow_{r_i.2.1} (3, w_1AP_i^1P_i^2P_i^3Cw_2) \Rrightarrow_{r_i.3.2}\\ &(2, w_1AP_i^1P_i^2P_i^3CCw_2) \Rrightarrow_{r_i.2.1, r_i.3.2}\\ &\qquad (2, w_1AP_i^1P_i^2P_i^2P_i^3CCCw_2) \Rrightarrow_{r_i.2.1, r_i.3.2} \dots \qquad (5.1)\end{aligned}$$

Since between symbols P_i^1 and P_i^3 there is at least one symbol P_i^2, there is no possibility to apply rule $r_i.2.2 : (\varepsilon, P_i^1P_i^3, \varepsilon; 1)_e$ and go to the first component. So, this computation can be omitted since it does not affect the final result.

Let us consider production $i : A \to BC$, where $A, B, C \in N$. The simulation of this rule is controlled by symbols P_i^1, P_i^2 and P_i^3. We also assume that the sentential form in the first component does not contain symbols from $\overline{P}$. Consider a string w_1Aw_2 in the first component, where $w_1, w_2 \in V^*$. By applying rules from Π that correspond to i-th production we obtain:

$$\begin{aligned}(1, w_1Aw_2) &\Rrightarrow_{r_i.1.1} (2, w_1AP_i^1w_2) \Rrightarrow_{r_i.2.1} (3, w_1AP_i^1P_i^2w_2) \Rrightarrow_{r_i.3.1}\\ &(4, w_1AP_i^1BP_i^2w_2) \Rrightarrow_{r_i.4.1} (4, w_1BP_i^2w_2) \Rrightarrow_{r_i.5.1} (4, w_1BP_i^2P_i^3w_2) \Rrightarrow_{r_i.4.2}\\ &\quad(4, w_1BP_i^2P_i^3Cw_2) \Rrightarrow_{r_i.3.2} (4, w_1BP_i^2Cw_2) \Rrightarrow_{r_i.2.2} (4, w_1BP_i^2P_i^3Cw_2)\end{aligned}$$

The simulation of production $i: A \to \alpha$, where $A \in N$ and $\alpha \in N \cup T$ is done in an analogous manner. Indeed, let $(1, w_1Aw_2)$ be a sentential form.

$$\begin{aligned}(1, w_1Aw_2) &\Rrightarrow_{r_i.1.1} (2, w_1AP_i^1w_2) \Rrightarrow_{r_i.2.1}\\ &(3, w_1AP_i^1\alpha w_2) \Rrightarrow_{r_i.3.1} (4, w_1AP_i^1P_i^2\alpha w_2) \Rrightarrow_{r_i.4.1}\\ &\quad(3, w_1P_i^2\alpha w_2) \Rrightarrow_{r_i.3.2} (2, w_1P_i^2P_i^3\alpha w_2) \Rrightarrow_{r_i.2.2} (1, w_1\alpha w_2)\end{aligned}$$

To finish the proof we observe that every sentential form has at most one symbol P_i^1, P_i^2 or P_i^3, $i = 1, \dots, n$. Moreover, the computations for $i : A \to BC$ and $i : A \to \alpha$ are deterministic, and additional computation of the form (5.1) for productions $i : AB \to AC$ does not produce new words.

As soon as P_i^1 is inserted in the first component either all rules corresponding to i-th rule have to be applied (in the defined order), or the derivation is blocked. Therefore, there is one-to-one correspondence between sentential forms (in the first component) and sentential forms of G. Hence, we have $L(G) = L(\Pi_1)$. □

The symmetrical case of the right context insertion gives

Corollary 5.2.8. *$ELSP_5(ins_1^{0,1}, del_2^{0,0}) = RE$.*

By a similar simulation we obtain the characterization of RE by exchanging the size of insertion and deletion rules.

Theorem 5.2.9. *$ELSP_5(ins_2^{0,0}, del_1^{1,0}) = ELSP_5(ins_2^{0,0}, del_1^{0,1}) = RE$.*

Proof. As in the previous theorem, we prove the inclusion $RE \subseteq ELSP_5(ins_2^{0,0}del_1^{1,0})$ by simulation of type-0 grammar in Penttonen normal form. The reverse inclusion follows from the Church thesis. Let $G = (N, T, S, P)$ be a type-0 grammar in Penttonen normal form. Suppose that rules in P are ordered and $n = Card(R)$.

Now consider the following system.
$\Pi = (V, T, \{S\}, R_1, R_2, R_3, R_4, R_5)$, where $V = N \cup T \cup \overline{P}$, $\overline{P} = \{P_i^j | i = 1, \dots, n,\ j = 1, \dots, 5\}$.

- For every context sensitive production $i : AB \to AC \in P$, where $A, B, C \in N$ consider:

$$\begin{aligned}
R_1^i =&\{r_i.1.1 : (\varepsilon, P_i^1 P_i^2, \varepsilon; 2)_a\}; \\
R_2^i =&\{r_i.2.1 : (P_i^2, B, \varepsilon; 3)_e, \qquad r_i.2.2 : (A, P_i^3, \varepsilon; 1)_e\}; \\
R_3^i =&\{r_i.3.1 : (\varepsilon, P_i^3 C, \varepsilon; 4)_a, \qquad r_i.3.2 : (A, P_i^2, \varepsilon; 2)_e\}; \\
R_4^i =&\{r_i.4.1 : (A, P_i^1, \varepsilon; 3)_e\}; \qquad R_5^i = \emptyset.
\end{aligned}$$

- For every context-free production $i : A \to BC$ from P with $A, B, C \in N$ consider following rules:

$$\begin{aligned}
R_1^i =&\{r_i.1.1 : (\varepsilon, P_i^1 P_i^2, \varepsilon; 2)_a\}; \\
R_2^i =&\{r_i.2.1 : (P_i^2, A, \varepsilon; 3)_e, \qquad r_i.2.2 : (\varepsilon, P_i^3, \varepsilon; 1)_e\}; \\
R_3^i =&\{r_i.3.1 : (\varepsilon, B P_i^3, \varepsilon; 4)_a, \qquad r_i.3.2 : (P_i^3, P_i^2, \varepsilon; 2)_e\}; \\
R_4^i =&\{r_i.4.1 : (P_i^3, P_i^1, \varepsilon; 5)_e, \qquad r_i.4.2 : (P_i^2, P_i^4, \varepsilon; 3)_e\}; \\
R_5^i =&\{r_i.5.1 : (\varepsilon, P_i^4 C, \varepsilon; 4)_a\}.
\end{aligned}$$

- For every production $i : A \to \alpha$ from P with $A \in N, \alpha \in N \cup T$ consider following rules:

$$\begin{aligned}
R_1^i =&\{r_i.1.1 : (\varepsilon, \alpha P_i^3, \varepsilon; 2)_a\}; \\
R_2^i =&\{r_i.2.1 : (P_i^3, A, \varepsilon; 3)_e, \qquad r_i.2.2 : (\alpha, P_i^2, \varepsilon; 1)_e\}; \\
R_3^i =&\{r_i.3.1 : (\varepsilon, P_i^1 P_i^2, \varepsilon; 4)_a \qquad r_i.3.2 : (\alpha, P_i^1, \varepsilon; 2)_e\}; \\
R_4^i =&\{r_i.4.1 : (\alpha, P_i^3, \varepsilon; 3)_e\}; \qquad R_5^i = \emptyset.
\end{aligned}$$

For every $i = 1, \dots, 5$ consider the following set of rules $R_1^i \cup \{r_i.1.1 : (\varepsilon, A, \varepsilon; 1)_e \mid i : A \to \varepsilon \in P\}$.

Now we claim that Π_2 generates the same language as G. We show that every step in derivation by grammar G can be simulated in Π_2. As in the previous theorem, we assume that the sentential form in the first component does not contain symbols from $\overline{P}$.

Let us consider production $i : AB \to AC \in R$. The simulation of this rule is controlled by symbols P_i^1, P_i^2 and P_i^3. Hence, the derivation by using the rules above is the following:

$$\begin{aligned}(1, w_1ABw_2) &\Rrightarrow_{r_i.1.1} (2, w_1AP_i^1P_i^2Bw_2) \Rrightarrow_{r_i.2.1}\\ &(3, w_1AP_i^1P_i^2w_2) \Rrightarrow_{r_i.3.1} (4, w_1AP_i^1P_i^2P_i^3Cw_2) \Rrightarrow_{r_i.4.1}\\ &\quad(3, w_1AP_i^2P_i^3Cw_2) \Rrightarrow_{r_i.3.2} (2, w_1AP_i^3Cw_2) \Rrightarrow_{r_i.2.2} (1, w_1ACw_2).\end{aligned}$$

Clearly, the first insertion $P_i^1P_i^2$ must be done between A and B, and the insertion of P_i^3C must be done adjacently right from P_2, otherwise rules $r_i.2.1$, $r_i.4.1$, and $r_i.2.2$ cannot be applied, and the derivation is blocked. We remark that for $(3, w_1AP_i^2P_i^3Cw_2)$ there is another rule that can be applied. Indeed, at this point it is possible to repeat the insertion $(\varepsilon, P_i^3C, \varepsilon; 4)_a$. In this case the derivation will be blocked in the next step as the form does not contain more symbols P_i^1, and hence no rule can be applied at the forth component. So, we simulate production $i : AB \to AC$.

Now we consider a context-free production $i : A \to BC$, where $A, B, C \in N$.

The simulation of the rule is controlled by symbols P_i^1, P_i^2, P_i^3 and P_i^4. The possible derivation is the following:

$$\begin{aligned}(1, w_1Aw_2) &\Rrightarrow_{r_i.1.1} (2, w_1P_i^1P_i^2Aw_2) \Rrightarrow_{r_i.2.1} (3, w_1P_i^1P_i^2w_2) \Rrightarrow_{r_i.3.1}\\ (4, w_1BP_i^3P_i^1P_i^2w_2) &\Rrightarrow_{r_i.4.1} (5, w_1BP_i^3P_i^2w_2) \Rrightarrow_{r_i.5.1} (4, w_1BP_i^3P_i^2P_i^4Cw_2) \Rrightarrow_{r_i.4.2}\\ &(3, w_1BP_i^3P_i^2Cw_2) \Rrightarrow_{r_i.3.2} (2, w_1BP_i^3Cw_2) \Rrightarrow_{r_i.2.2} (1, w_1BCw_2).\end{aligned}$$

It is clear that the first insertion of $P_i^1P_i^2$ must be performed adjacently left from A, because of the rule $r_i.2.1$, and the insertion of BP_i^3 must be performed adjacently left from P_i^1 because of the rule $r_i.4.1$. We mention that for sentential form $(3, w_1BP_i^3P_i^2Cw_2)$ in addition to the derivation shown above, the context free insertion of BP_i^3 by the rule $r_i.3.1$ can be performed. In this case the derivation will be blocked in component 4, as no rules may be applied to the string. So, we simulate rule $i : A \to BC$ correctly.

Now, consider production $i : A \to \alpha \in R$, where $A \in N, \alpha \in N \cup T$. The derivation for this case has the following form:

$$(1, w_1Aw_2) \Rrightarrow_{r_i.1.1} (2, w_1\alpha P_i^3 Aw_2) \Rrightarrow_{r_i.2.1} (3, w_1\alpha P_i^3 w_2) \Rrightarrow_{r_i.3.1} (4, w_1\alpha P_i^3 P_i^1 P_i^2 w_2) \Rrightarrow_{r_i.4.1} (3, w_1\alpha P_i^1 P_i^2 w_2) \Rrightarrow_{r_i.3.2} (2, w_1\alpha P_i^2 w_2) \Rrightarrow_{r_i.2.2} (1, w_1\alpha w_2)$$

This case of replacement basically uses one insertion αP_i^3 adjacently left from A, and two deletion rules $(P_i^3, A, \varepsilon; 3)_e$ and $(\alpha, P_i^3, \varepsilon; 3)_e$. Here we introduce one additional insertion rule $(\varepsilon, P_i^1 P_i^2, \varepsilon; 4)_a$ and two deletion rules $(\alpha, P_i^1, \varepsilon; 2)_e$, $(\alpha, P_i^2, \varepsilon; 1)_e$ in order to have an even number of insertion and deletion rules for the production. So, we simulate rule $i : A \to \alpha$ correctly.

Finally, every production $i : A \to \varepsilon$, $A \in N$ is simulated directly in the first component by the corresponding rule $r_i.1.1 : (\varepsilon, A, \varepsilon; 1)_e$.

To claim the proof we observe that every correct sentential form preserves the following property: no symbol from $\overline{P}$ is present in the first component. As shown before the insertion of $P_i^1 P_i^2$ or P_i^3 for the corresponding $i - th$ rule in the first component results to either all rules corresponding to i-th production have to be applied (in the defined order) or the derivation is blocked. Hence, we have $L(G) = L(\Pi_2)$. □

The next theorem shows that graph-control does not always increase the computational power to the maximum.

Theorem 5.2.10. *$REG \setminus ELStP_*(ins_2^{0,0} del_2^{0,0}) \neq \emptyset$.*

Proof. We show that $L_{ab} = a^*b \notin ELStP_k(ins_2^{0,0} del_2^{0,0})$, for any $k \geq 1$. Let Π be a graph-controlled insertion-deletion system having context-free rules that may insert or delete at most two symbols and that $L(\Pi) = L_{ab}$. We prove the assertion of the theorem by contradiction.

We construct a partition of rules $P_0 \cup P_1 \cup \cdots \cup P_r, r \geq 1$ for an arbitrary finite derivation in Π (of sufficient length) $\Pi : w_0 \Rightarrow^* w$ such that the overall effect of rules from each $P_i, i = 1, \ldots, r', r' \leq r$ is the context-free insertion of at most two terminals. Moreover, $r - r' \leq k$, $k = |w_0|$, assuming, the length of w is sufficiently large.

Indeed, for i-th letter of w consider those rules that "contribute" to this letter and denote this by P'_i. Hence, we obtain n sets of rules P'_i, where $1 \leq i \leq n$, and $n = |w|$ is the length of the word. Next, we eliminate those P'_i, having $P'_i = \emptyset$, that correspond to the letters from the axiom w_0. Since, P'_i contribute to at most two letters(see [67]), we may consider as desired elements of the partition $P_i, 1 \leq i \leq r$ only those P'_i which are not duplicated. Finally, we add to the list the set of all rules P_0 of the derivation that contribute to empty words.

We mention that the construction of the partition is similar to the one from [67], where pure insertion-deletion systems of the size $(2, 0, 0; 2, 0, 0)$ are studied. The difference is that in the case of graph-controlled insertion-deletion systems we need to preserve the information about the component in which corresponding rules are applied. Moreover, P_0 can be omitted for pure insertion-deletion systems, while in the case of graph-controlled insertion-deletion systems P_0 defines a path through some components.

Assuming that w is of sufficient length, it is clear that some applications of rules from $P_i, i = 1, \ldots, r'$ which insert a or aa should be performed. Since the insertion is context-free, such an application can happen at the end of the word leading to a word having a preceded by b, which is a contradiction. $\square$

However, Example 5.1.2 shows the strict inclusion $INS_2^{0,0}DEL_2^{0,0} \subset ELStP_3(ins_2^{0,0}del_2^{0,0})$. In Table 5.1 we summarize the obtained results about generating power of graph-controlled insertion-deletion systems.

Table 5.1: Complexity classes for the minimal graph-controlled insertion-deletion systems

$ins_n^{m,m'} \backslash del_p^{q,q'}$	1,0,0	1,1,0 and 1,0,1	2,0,0
1,0,0	$\subseteq PsMAT^\lambda$		
1,1,0 and 1,0,1		RE	RE
2,0,0	$\subset MAT^\lambda$	RE	incomparable with Reg

Chapter 6

Graph-controlled insertion-deletion systems with priorities

This chapter is devoted to graph-controlled insertion-deletion systems with priorities. This model has the following property: if a component contains an insertion and a deletion rule being both applicable, then the deletion is always chosen to be applied. Saying otherwise, no insertion rule can be applied if a deletion rule is applicable. We show that such systems are a particular case of graph-controlled insertion-deletion systems with appearance checking. The results on computational complexity are given for the classes of languages which can be generated by such systems of very small sizes.

The family of languages corresponding to graph-controlled insertion-deletion systems with priorities is denoted by $(E)LSP_k(ins_n^{m,m'} < del_p^{q,q'})$. As in the previous section the letter t is inserted before P to denote classes whose communication graph is arbitrary, e.g., $ELStP_k(ins_n^{m,m'} < del_p^{q,q'})$.

6.1 Appearance checking and priorities

The introduced priority of deletion over insertion can be considered as a particular case of appearance checking. Indeed, one can interpret the priority of deletions over insertions as follows: if a site that corresponds to a deletion rule is presented then the deletion is performed, otherwise (by appearance checking) any insertion rule of the same component can be performed. In other words, for a system with the priority one can easily construct an equal system working in appearance checking mode.

Lemma 6.1.1. *For any $L \in ELStP_k(ins_n^{m,m'} < del_p^{q,q'}), k \geq 1, n, m, m', p, q, q' \geq 0$ there is graph-controlled insertion-deletion system Π working in appearance checking mode, such that $L = L(\Pi)$, having the same size and using $2k$ components.*

Proof. Let $\Pi' = (V, T, A, i_0, i_f, R_1, \ldots, R_k)$ be a graph-controlled insertion-deletion system with priorities such that $L = L(\Pi')$. Let $R_i, i = 1, \ldots, k$ be a set of rules corresponding to i-th component of Π'. Denote by

$$\Pi = (V, T, i_{0'}, i_{\overline{f}}, R_{\overline{f}}, R_{1'}, R_{1''}, \ldots, R_{k'}, R_{k''}),$$

the following graph-controlled insertion-deletion system working in appearance checking mode:

$$R_{i'} = \{(u, v, w; j', i'')_e \mid (u, v, w; j)_e \in R_i\}, i = 1, \ldots, k,$$

$$R_{i''} = \{(u, v, w; j', i'')_a \mid (u, v, w; j)_a \in R_i\}, i = 1, \ldots, k,$$

$$R_{\overline{f}} = \emptyset.$$

For every rule $(u, v, w; i_f)_e$ from i-th component we add to $R_{i'}$ rule $(u, v, w; i_{\overline{f}}, i'')_e$, also, for every rule $(u, v, w; i_f)_a$ from i-th component we add to $R_{i''}$ rule $(u, v, w; i_{\overline{f}}, i'')_a$.

We claim that $L(\Pi) = L(\Pi')$. Indeed, every component of Π' is simulated by two new components, such that the first one performs every deletion (if any possible), and the second component performs corresponding insertions. The final component $R_{\overline{f}}$ collects the resulting strings. □

The correspondence between graph-controlled insertion-deletion systems working in appearance checking mode and in the mode of priorities of deletion over insertion is illustrated by following example.

Example 6.1.2. *Let $T = \{a\}$ be a terminal alphabet and $V = \{a, A\}$ be a working alphabet. Consider the graph-controlled scheme working in appearance checking mode*

$$\Pi = (V, T, \{\varepsilon\}, 0, 3, R_0, R_1, R_2, R_3),$$

where

$$R_0 = \{r_0 : (A, a, A; 0, 1)_a\}, \qquad R_1 = \{r_1 : (\varepsilon, A, \varepsilon; 1, 2)_e\},$$
$$R_2 = \{r_2 : (\varepsilon, a, \varepsilon; 3, 2)_a\}, \qquad R_3 = \{r_3 : (a, AA, a; 3, 0)_a\}.$$

The system Π_2 generates the nonsemilinear language $\{a^{2^n} \mid n \geq 0\}$. Indeed, starting

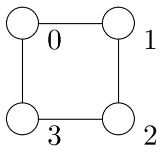


Figure 6.1: Graph structure for Example 6.1.2.

from the axiom ε we have the following derivation

$$(0, \varepsilon) \Rrightarrow_{r_0} (1, \varepsilon) \Rrightarrow_{r_1} (2, \varepsilon) \Rrightarrow_{r_2} (3, a).$$

By the next round aa appears in the final component:

$$(3, a) \Rrightarrow_{r_3} (0, a) \Rrightarrow_{r_0} (1, a) \Rrightarrow_{r_1} (2, a) \Rrightarrow_{r_2} (3, aa).$$

In general, for every configuration $(3, w), w \in (aAA + a)^+a$ the rule $(a, AA, a; 3, 0)_a$ can be applied until every pair of a is separated by AA. At this moment by the appearance checking the word of the form $(aAA)^+a$ is sent to component 0. By the same principle in component 0 every pair of A's is separated by insertions of a's. Hence, we have the following derivation:

$$(3, a^k) \Rrightarrow^*_{r_3} (0, (aAA)^{k-1}a) \Rrightarrow^*_{r_0} (1, (aAaA)^{k-1}a) \Rrightarrow^*_{r_1} (2, a^{2(k-1)}a) \Rrightarrow^*_{r_2} (3, a^{2k}).$$

By iterating the above sequence it is possible to generate every word of the lengths of powers 2. Considering all output strings from component 3, *we get* $L(\Pi) = \{a^{2^n} \mid n \geq 0\}$.

It is also possible to construct a system with priorities, if a system working in appearance checking mode is given. We illustrate it by modifying the example above. Consider the component R_0 of the above example with rule r_0 working in appearance checking mode. This component can be replaced by three new components $R_{0'}, R_{0''}, R_{\overline{0}}$ working in the mode with priorities.

$$R_{0'} = \{r_{0'} : (A, a, A; 0')_a; r_{0'.1} : (\varepsilon, \varepsilon, \varepsilon; 0'')_a\},$$

$$R_{0''} = \{r_{0''} : (\varepsilon, AA, \varepsilon; \overline{0})_e; r_{0''.1} : (\varepsilon, \varepsilon, \varepsilon; 1)_a\},$$

where $R_{\overline{0}}$ is a component with no rules. The rules $r_{0'}$ and $r_{0''}$ simulate the appearance checking of AA (like in the rule r_0), while rules $r_{0'.1}$ and $r_{0''.1}$ provide the transitions when maximal possible number of insertion of a have been performed. Clearly, this idea can be generalized on arbitrary graph-controlled insertion-deletion system working in appearance checking mode. In the next sections of the chapter we consider only models with priorities.

6.2 Context-freeness with priorities

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

Theorem 6.2.1. *$REG \backslash LStP_*(ins_n^{0,0} < del_1^{0,0}) \neq \emptyset$, for any $n > 0$.*

However, in terms of Parikh sets these systems are computationally complete. We now give a more sophisticated proof for the tree-like component structure.

Theorem 6.2.2. *$PsSP_*(ins_1^{0,0} < del_1^{0,0}) = PsRE$.*

Proof. The proof is done by showing that for any register machine $\mathcal{M} = (d, Q, q_0, h, P)$ there is a system $\Pi \in PsSP_*(ins_1^{0,0} < del_1^{0,0})$ with $Ps(\mathcal{M}) \subseteq Ps(\Pi)$. Then the existence of register machines generating $PsRE$ implies $PsRE \subseteq Ps(\Pi)$. The converse inclusion follows from the Church-Turing thesis.

Let Q_+ (Q_-) be the sets of labels of increment (conditional decrement, respectively) instructions of a register machine, and let $Q = Q_+ \cup Q_- \cup \{h\}$ represent all instructions. Consider a graph-controlled system with alphabet $Q \cup \{A_i \mid 1 \le i \le d\} \cup \{Y\}$ and the component structure illustrated in Figure 6.2. (The structures in the dashed rectangles are repeated for every instruction of the register machine.)

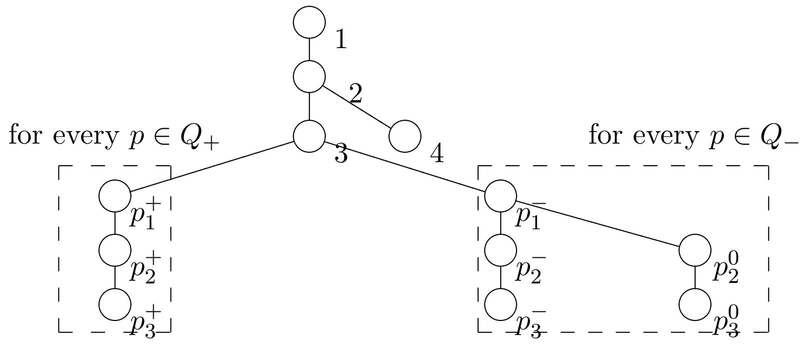


Figure 6.2: Graph structure for Theorem 6.2.2

Initially there is a single string q_0 in component 3. The rules are the following.

$$
\begin{aligned}
R_1 = \{ \quad & 1 : (\varepsilon, Y, \varepsilon; out)_e\}, & & \\
R_2 = \{ \quad & 2.1 : (\varepsilon, Y, \varepsilon; out)_a, & & 2.2 : (\varepsilon, Y, \varepsilon; in_4)_e\}, \\
R_3 = \{ \quad & 3.1 : (\varepsilon, p, \varepsilon; in_{p_1^+})_e \mid p \in Q_+\} \cup & & \{3.2 : (\varepsilon, p, \varepsilon; in_{p_1^-})_e \mid p \in Q_-\} \\
\cup \{ \quad & 3.3 : (\varepsilon, Y, \varepsilon; here)_e, & & 3.4 : (\varepsilon, h, \varepsilon; out)_e\},
\end{aligned}
$$

For any rule $p : (ADD(k), q, s)$, $R_{p_3^+} = \emptyset$ and

$$
\begin{aligned}
R_{p_1^+} = \{ \quad & a.1.1 : (\varepsilon, A_k, \varepsilon; in_{p_2^+})_a, & & a.1.2 : (\varepsilon, Y, \varepsilon; out)_a\}, \\
R_{p_2^+} = \{ \quad & a.2.1 : (\varepsilon, q, \varepsilon; out)_a, & & a.2.1' : (\varepsilon, s, \varepsilon; out)_a, \\
& a.2.2 : (\varepsilon, q, \varepsilon; in_{p_3^+})_e, & & a.2.2' : (\varepsilon, s, \varepsilon; in_{p_3^+})_e\},
\end{aligned}
$$

For any rule $p : (SUB(k), q, s)$, $R_{p_3^-} = R_{p_3^0} = \emptyset$ and

$$\begin{aligned}
R_{p_1^-} = \{ \quad & e.1.1 : (\varepsilon, A_k, \varepsilon; in_{p_2^-})_e, && e.1.2 : (\varepsilon, Y, \varepsilon; in_{p_2^0})_a, && e.1.3 : (\varepsilon, Y, \varepsilon; out)_e\},\\
R_{p_2^-} = \{ \quad & e.2.1 : (\varepsilon, q, \varepsilon; out)_a, && e.2.2 : (\varepsilon, q, \varepsilon; in_{p_3^-})_e, &&\\
& e.2.3 : (\varepsilon, s, \varepsilon; in_{p_3^-})_e, && e.2.4 : (\varepsilon, Y, \varepsilon; here)_a\}, &&\\
R_{p_2^0} = \{ \quad & e.3.1 : (\varepsilon, s, \varepsilon; out)_a, && e.3.2 : (\varepsilon, q, \varepsilon; in_{p_3^0})_e, && e.3.3 : (\varepsilon, s, \varepsilon; in_{p_3^0})_e\}.
\end{aligned}$$

In component 3 configurations $(p, x_1, \dots, x_n)$ of $\mathcal{M}$ are encoded by strings $Perm(pA_1^{x_1} \dots A_n^{x_n} Y^t)$, $t \geq 0$.

We say that such strings have a *simulating* form. Clearly, in the initial configuration the string is already in the simulating form.

To prove that system Π correctly simulates $\mathcal{M}$ we prove the following claims:

1. For any transition $(p, x_1 \dots x_n) \Rightarrow (q, x_1', \dots, x_n')$ in $\mathcal{M}$ there exist a computation in Π from the configuration containing $Perm(pA_1^{x_1} \dots A_n^{x_n} Y^t)$ in component 3 to the configuration containing $Perm(qA_1^{x_1'} \dots A_n^{x_n'} Y^{t'})$, $t' \geq 0$ in component 3 such that during this computation component 3 is empty on all intermediate steps and, moreover, this computation is unique.

2. For any successful computation in Π (yielding a non-empty result), componenet 3 contains only strings of the above form.

3. The result $(x_1, \dots, x_n)$ in Π is obtained if and only if a string of form $Perm(hA_1^{x_1} \dots A_n^{x_n})$ appears in component 3.

Now we prove each claim from above. Consider a string $Perm(pA_1^{x_1} \dots A_n^{x_n} Y^t)$, $t \geq 0$ in component 3 of Π. Take an instruction $p : (ADD(k), q, s) \in P$. The only applicable rule in Π is from the group 3.1 (in the future we simply say rule 3.1) yielding the string $Perm(A_1^{x_1} \dots A_n^{x_n} Y^t)$ in component p_1^+. After that rule $a.1.1$ is applied yielding string $Perm(A_1^{x_1} \dots A_k^{x_k+1} \dots A_n^{x_n} Y^t)$ in component p_2^+. After that one of rules $a.2.1$ or $a.2.1'$ is applied; then rule $a.1.2$ yields one of strings $Perm(zA_1^{x_1} \dots A_k^{x_k+1} \dots A_n^{x_n} Y^{t+1})$, $z \in \{q, s\}$, is in the simulating form.

Now suppose that there is an instruction $p : (SUB(k), q, s) \in P$. Then the only applicable rule in Π is 3.2 which yields the string $Perm(A_1^{x_1} \dots A_n^{x_n} Y^t)$ in component p_1^-. Now if $x_k > 0$, then, due to the priority, rule $e.1.1$ will be applied followed by application of rules $e.2.4$, $e.2.1$ and $e.1.3$ which yields the string $Perm(qA_1^{x_1} \dots A_k^{x_k-1} \dots A_n^{x_n} Y^{t'})$ that is in the simulating form. If $x_k = 0$, then rule $e.1.2$ will be applied (provided that all symbols Y were previously deleted by rule 3.3), followed by rules $e.3.1$ and $e.1.3$ which leads to the string $Perm(sA_1^{x_1} \dots A_n^{x_n})$ that is in the simulating form.

To show that component 3 is empty during the intermediate steps, we prove the following invariant:

Invariant 1. During a successful computation, any visited component p_1^+ or p_1^- is visited an even number of times as follows: first a string coming from component 3 is sent to an inner component (p_2^+, p_2^- or p_2^0) and after that a string coming from an inner component is sent to component 3.

Indeed, since there is only one string in the initial configuration, it is enough to follow only its evolution. Hence, a sting may visit the node p_1^+ or p_1^- only if on the previous step symbol p was deleted by one of rules 3.1 or 3.2. If one of rules $a.1.2$ or $e.1.3$ is applied, then component 3 will contain a string of form $Perm(A_1^{x_1} \dots A_n^{x_n} Y^t)$ which cannot evolve anymore because all rules in component 3 imply the presence of symbol from the set Q. Hence, the string is sent to an inner component. In the next step the string will return from the inner component by one of rules $a.2.1$, $a.2.1'$, $e.2.1$ or $e.3.1$ inserting a symbol from Q. If the string enters an inner component again, then it will be sent to a trap component (p_3^+, p_3^- or p_3^0) by rules deleting symbols from Q. Hence the only possibility is to go to component 3 (a string that visited component p_2^- will additionally use rule $e.2.4$).

For the second claim, it suffices to observe that the invariant above ensures that in component 3 only one symbol from Q can be present in the string.

The third claim holds since a string may move to component 2 if and only if the final label h of $\mathcal{M}$ appears in component 3. Then, the string is checked for the absence of symbols Y by rule 2.2 (note that symbols Y can be erased in component 3 by rule 3.3) and sent to the environment by rules 2.1 and 1. Hence, the string sent to the environment will contain only occurrences of symbols A_i, $1 \leq i \leq d$.

By induction on the number of computational steps we obtain that Π simulates any computation in $\mathcal{M}$. Claim 1 and 2 imply it is not possible to generate other strings and Claim 3 implies that the same result is obtained. □

We remark that an empty string may be obtained during the proof. This string can still evolve using insertion rules. If we would like to forbid such evolutions, it suffices to use a new symbol, e.g. X, in the initial configuration, add new surrounding component and a rule that deletes X from it.

6.3 Small contextual systems with priorities

Although Theorem 6.2.2 shows that the systems from the previous section are quite powerful, they cannot generate RE without control on the place where a symbol is inserted. Once we allow a context in insertion or deletion rules, they can.

Theorem 6.3.1. *$LSP_*(ins_1^{0,1} < del_1^{0,0}) = RE$.*

Proof. We simulate a register machine with WRITE instructions. We implement this instruction as an ADD instruction, except the added symbol has to be inserted to the left of a special marker, deleted at the end, as follows:

- Replace any writing instruction $p : (WRITE(A), q, s)$, $A \in T$ of the machine by instructions $p : (ADD(A), q, s)$, considering output symbols A like new dummy register. Construct the system Π as in Theorem 6.2.2.
- Change the initial string in component 3 to q_0M;
- Replace rules $a.1.1$ $((\varepsilon, A, \varepsilon; in_{p_2^+})_a \in R_{p_1^+})$ by $(\varepsilon, A, M; in_{p_2^+})_a$ for $A \in T$;
- Surround component 1 by a new component o and add to it the following rule $R_o = \{(\varepsilon, M, \varepsilon; out)_e\}$.

It is easy to see that the above construction permits to correctly simulate the register machine with writing instructions. □

Taking M in the left context yields the mirror language. Since RE is closed with respect to the mirror operation we get the following corollary:

Corollary 6.3.2. $LSP_*(ins_1^{1,0} < del_1^{0,0}) = RE.$

A similar result holds if contextual operation of deletion is allowed.

Theorem 6.3.3. $LSP_*(ins_1^{0,0} < del_1^{1,0}) = RE.$

Proof. As in Theorem 6.3.1, we use the construction from Theorem 6.2.2. However, an additional component structure is needed to simulate the writing instructions.

We modify the construction of Theorem 6.2.2 as follows. Let Q_s be the set of labels of WRITE instructions of a register machine. We add following substructures $\mu_{\langle ps \rangle}$ inside component 3 (shown in Figure 6.3).

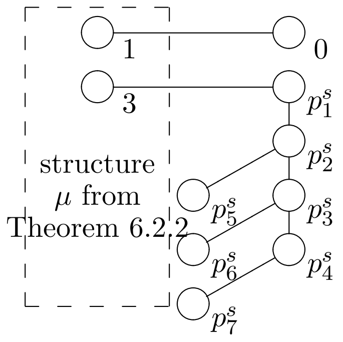


Figure 6.3: Component structure for Theorem 6.3.3.

As in Theorem 6.2.2 the initial configuration contains a single string q_0 in component 3. The system contains sets of rules R_1, R_2, $R_{p_1^+}, R_{p_2^+}, R_{p_3^+}$, $R_{p_1^-}, R_{p_2^-}, R_{p_3^-}$, $R_{p_2^0}, R_{p_3^0}$ defined as in Theorem 6.2.2. There are also following additional rules for instructions $p : (WRITE(A), q)$ (the rule set R_3' shall be added to R_3).

$$
\begin{aligned}
R_0 &= \{ && 0 :(\varepsilon, M, \varepsilon; out)_e\} \\
R_3' &= \{ && 3.5 :(\varepsilon, p, \varepsilon; in_{p_1^s})_e \mid p \in Q_s\}, \\
R_{p_1^s} &= \{ && w.1.1 :(\varepsilon, M, \varepsilon; in_{p_2^s})_a, && w.1.2 :(\varepsilon, M, \varepsilon; out)_e\}, \\
R_{p_2^s} &= \{ && w.2.1 :(\varepsilon, M', \varepsilon; in_{p_3^s})_a, && w.2.2 :(\varepsilon, M', \varepsilon; out)_e\} \\
&\cup \{ && w.2.3 :(M, x, \varepsilon; in_{p_5^s})_e \mid x \in O\}, \\
R_{p_3^s} &= \{ && w.3.1 :(\varepsilon, A, \varepsilon; in_{p_4^s})_a, && w.3.2 :(\varepsilon, Y, \varepsilon; out)_a\} \\
&\cup \{ && w.3.3 :(x, M, \varepsilon; in_{p_6^s})_e \mid x \in O \setminus \{M', q\}\}, \\
R_{p_4^s} &= \{ && w.4.1 :(\varepsilon, q, \varepsilon; out)_a, && w.4.2 :(M', M, \varepsilon; in_{p_7^s})_e\}, \\
R_{p_5^s} &= \emptyset, && R_{p_6^s} = \emptyset,\ R_{p_7^s} = \emptyset.
\end{aligned}
$$

We simulate the WRITE instruction as follows. Suppose the configuration of register machine is $pA_1^{x_1}\dots A_d^{x_d}$ and the word $a_1\dots a_n$ is written on the output tape. The corresponding simulating string in Π will be of form $p \sqcup\!\sqcup\, w$, where $w = Perm(A_1^{x_1}\dots A_d^{x_d}Y^t) \sqcup\!\sqcup\, a_1\dots a_n$, $t \geq 0$. After the deletion of the state symbol p, a marker M is inserted in the string by rule $w.1.1$. If M is not inserted at the right end of the string, in the next step rule $w.2.3$ is applicable and the string enters the trap component p_5^s. In the next step symbol M' is inserted in the string. If it is not inserted before M, then the string is sent to component p_6^s by rule $w.3.3$. Hence, at this moment the contents of component p_3^s is $wM'M$. If rule $w.3.2$ is used, then the string $Y \sqcup\!\sqcup\, w$ reaches component 3 and no rule is applicable anymore. Otherwise, symbol A is inserted by rule $w.3.1$. If it is not between M' and M, then rule $w.4.2$ is applicable and the string enters component p_7^s. After that q is inserted between A and M, otherwise the trapping rule $w.3.3$ is applicable. At this moment, the configuration of the system consists of the string $w_tM'AqM$ in component p_3^s. Now if the rule $w.3.1$ is used, then the string is sent to the trap component by rule $w.4.1$. Otherwise, rule $w.3.2$ should be used followed by the application of rules $w.2.2$ and $w.1.2$, leading to string $Y \sqcup\!\sqcup\, wAq$ in component 3. Hence, the symbol A is appended at the end of the string. At the end of the computation, all symbols Y are deleted when the string gets component 0 where the symbol M is further deleted and the string is sent out. Hence, all symbols from $O - T$ are deleted and a word generated by $\mathcal{M}$ is obtained.

The converse inclusion $LSP_*(ins_1^{0,0} < del_1^{1,0}) \subseteq RE$ can be obtained from the Church-Turing thesis. $\square$

Since RE is closed with respect to the mirror operation we obtain:

Corollary 6.3.4. *$LSP_*(ins_1^{0,0} < del_1^{0,1}) = RE$.*

We remark that the contextual deletion was used only to check for erroneous evolutions. Therefore we can replace it by a context-free deletion of two symbols.

Theorem 6.3.5. *$LSP_*(ins_1^{0,0} < del_2^{0,0}) = RE$.*

Proof. We modify the proof of Theorem 6.3.3 as follows.

- Replace rules $w.2.3$ $((M, x, \varepsilon; in_{p_5^s})_e \in R_{p_2^s})$ by rules $(\varepsilon, Mx, \varepsilon; in_{p_5^s})_e$,
- Replace rules $w.3.3$ $((x, M, \varepsilon; in_{p_6^s})_e \in R_{p_3^s})$ by rules $(\varepsilon, xM, \varepsilon; in_{p_6^s})_e$,
- Replace rules $w.4.2$ $((M', M, \varepsilon; in_{p_7^s})_e \in R_{p_4^s})$ by rules $(\varepsilon, M'M, \varepsilon; in_{p_7^s})_e$.

The role of the new rules is the same as the role of the rules that were replaced. More exactly, the system checks whether two certain symbols are consecutive and if this is the case, the string is blocked in a non-output component. $\square$

We mention that the counterpart of Theorem 6.3.5 obtained by interchanging parameters of the insertion and deletion rules is not true, see Theorem 6.2.1.

In Table 6.1 we summarize the obtained results about generaive power of graph-controlled insertion-deletion systems with priorities.

Table 6.1: Results on graph-controlled insertion-deletion systems *with* priorities

$ins_n^{m,m'} \backslash del_p^{q,q'}$	1,0,0	1,1,0 and 1,0,1	2,0,0
1,0,0	$PsRE$	RE	RE
1,1,0 and 1,0,1	RE	RE	RE
$n, 0, 0 \quad n \geq 2$	incomparable with Reg	RE	RE

Conclusions

In this thesis we have studied insertion-deletion systems and their extensions. We started our investigation from the pure insertion systems and their extensions by the graph-controlled scheme. We showed a series of characterizations of known language classes by the means of such systems.

Next we considered the generative power of insertion-deletion systems. We closed the last open question for symmetrical insertion-deletion systems concerning computational completeness of systems of size $(2, 0, 0; 1, 1, 1)$. After that we concentrated on the study of non-symmetrical insertion-deletion systems. We showed a series of completeness results for systems of small sizes. The summary of these results is given in Table 4.2. The most interesting is that we showed that some classes of insertion-deletion systems are not computationally complete. We also succeeded to give a characterization to one of them.

Further, for these uncomplete systems we introduced and studied graph-controlled variants. In all the cases we observed that the computational power of such systems is increased. Finally, we made another extension of graph-controlled systems by introducing priorities that are a particular case of appearance checking. This further increases the computational power. Surprisingly, we showed that it is possible to obtain the whole class $PsRE$ with the system having context-free one symbol insertion and deletion rules. By introducing contextual insertion and deletion, any RE language can be generated.

The obtained results permit us to approach the question about the borderline between computationally complete and uncomplete systems for insertion-deletion systems.

Now, let us formulate some interesting problems raised by our research. First of all we recall Table 4.2, where the results for one-sided insertion-deletion systems are collected. It follows from this table that there are several one-sided insertion-deletion systems for which the generative power is not yet known. Observing the table we conjecture that for systems of size $(n, m, m', 2, 0, 0)$, if $n + m + m' \geq 3$, we can get the computational completeness, while if $n+m+m' < 3$ the corresponding systems are uncomplete. Similarly, we conjecture that systems of size $(n, m, m', 1, 1, 0)$ are computationally complete, if $n + m + m' \geq 4$, while in the case $n + m + m' < 4$ such systems are likely to be uncomplete. The insertion-deletion systems which have both contexts (for insertion and for deletion) on the same side, i.e, of the form $(n, m, 0; p, q, 0), n + p \leq 4, m + q \geq 1$ need further investigation. We think that the method of direct simulation used in Chapter 4 gives a good way to solve these problems.

Table 6.2 summarizes the systems for which the question about their computational complexity has not been answered. Lines 1 and 2 of the table represent the systems having insertion rules with the maximum length of insertion strings at least two, while the systems corresponding to lines $3-8$ can insert at most one symbol. Lines $1-4$ are mirrored copies of lines $5-8$ with respect to insertion and deletion parts. We omit the cases which correspond to the interchanges of the left and right sizes of contexts.

Next problem investigated in the thesis refers to uncomplete insertion-deletion systems. In Chapter 4 we have studied one-sided uncomplete systems of sizes $(1, 1, 0; 1, 1, 0)$, $(1, 1, 0; 2, 0, 0)$ and $(2, 0, 0; 1, 1, 0)$. While we described the family $INS_1^{1,0}DEL_1^{1,0}$, for the latter two families we could only provide some examples of regular languages that cannot be generated by either of them. On the other hand, Lemmas 4.5.4 and 4.5.4 show that $INS_1^{1,0}DEL_1^{1,0}$ is a subclass of $INS_2^{0,0}DEL_1^{1,0} \cap INS_1^{1,0}DEL_2^{0,0}$. The question whether we could provide more precise characterization for these two small families of languages remains open.

In Chapter 5 uncomplete graph-controlled insertion systems of the size $(2, 0, 0; 2, 0, 0)$ were considered. Insertion-deletion systems of the same size generate the family of languages which is a proper subclass of context-free languages [67].

Table 6.2: Open problems for insertion-deletion systems with one-sided contexts

Nb.	$(n, m, m'; p, q, q')$	size
1	$(n, m, m'; 1, q, 0)$, $n \geq 2, m, m' \leq 1, q \geq 1$, $n + m + m' \geq 3$	$n + m + m' + q + 1$
2	$(n, m, 0; 1, q, 0), n, m \geq 2, q \geq 1$	$n + m + q + 2$
3	$(1, m, 1; 1, q, 0), m \geq 1, q \geq 2$	$m + q + 3$
4	$(1, m, 0; 1, 0, 1), m \geq 2$	$m + 3$
5	$(1, m, 0; p, q, q')$, $p \geq 2, q, q' \leq 1, m \geq 1$, $p + q + q' \geq 3$	$m + p + q + q' + 1$
6	$(1, m, 0; p, q, 0), p, q \geq 2, m \geq 1$	$m + p + q + 2$
7	$(1, m, 0; 1, q, 1), m \geq 1, q \geq 2$	$m + q + 3$
8	$(1, 0, 1; 1, q, 0), q \geq 2$	$q + 3$

For the graph-controlled case one would expect a similar inclusion in the family of matrix languages. It appears that the technique of elimination of deletion rules presented in [67] does not work for the graph-controlled case. So, the question about the upper bound for this family remains open.

Another problem concerning graph-controlled insertion-deletion systems is related to the number of used components. Indeed, the number of used components corresponds to the amount of meta−states with which it is possible to specify the system. Our results on the computational completeness were obtained with at most five components. We suggest that this number could be reduced to a smaller number. Indeed, looking closer in the proofs of Theorems 5.2.5, 5.2.7, 5.2.8 one could note that we have used several redundant symbols and in some cases redundant computations. Hence, it would be interesting to get the smallest possible bound for the number of used components.

We would like to mention that our study concentrates mostly on the *descriptional* complexity of the systems and on the modeling power corresponding to their size. While the descriptional complexity may answer the questions of the generative power, one may be interested in functioning of the system in "real time". Hence, it could be important to investigate such dynamical properties of the insertion-deletion systems as length of computations, frequency of visited components, confluence of the computation, etc.

Our investigation showed that the concept of insertion-deletion systems can be adapted for the graph-controlled framework. It can be also useful to extend insertion-deletion systems by other regulations as matrix, (un)ordered, random-context, conditional, valence grammars, etc.

In conclusion, we would like to add that the study of the insertion-deletion systems is quite inspiring and could give many new opportunities and we expect many outcomes in the future.

Bibliography

[1] L. M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, 11, 1994.

[2] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of the Cell, Fourth Edition.* Garland, 2002.

[3] A. Alhazov, E. Csuhaj-Varjú, C. Martín-Vide, and Y. Rogozhin. On the size of computationally complete hybrid networks of evolutionary processors. *Theor. Comput. Sci.*, 410(35):3188–3197, 2009.

[4] A. Alhazov, G. B. Enguix, and Y. Rogozhin. Obligatory hybrid networks of evolutionary processors. In J. Filipe, A. L. N. Fred, and B. Sharp, editors, *Proceedings of the International Conference on Agents and Artificial Intelligence, ICAART 2009, Porto, Portugal, January 19 - 21, 2009*, pages 613–618. INSTICC Press, 2009.

[5] A. Alhazov, R. Freund, and A. Riscos-Núñez. One and two polarizations, membrane creation and objects complexity in P systems. In *SYNASC*, volume Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2005), 25-29 September 2005, Timisoara, Romania, pages 385–394. IEEE Computer Society, 2005.

[6] A. Alhazov, A. Krassovitskiy, Y. Rogozhin, and S. Verlan. A note on P systems with small-size insertion and deletion. In *Proceedings of the 10th Workshop on Membr ane Computing, WMC10, Curtea de Arges (Romania), August 24-27*, pages 534–537, 2009.

[7] A. Alhazov, A. Krassovitskiy, Y. Rogozhin, and S. Verlan. P systems with minimal insertion and deletion. In *BWMC09*, pages 9–22, 2009.

[8] A. Alhazov, A. Krassovitskiy, Y. Rogozhin, and S. Verlan. *Small Size Insertion and Deletion Systems*, volume 2. Scientific Applicatioons of Language Methods of *Mathematics, Computing, Language, and Life: Frontiers in Mathematical Linguistics and Language Theory*, pages 459–524. World Scientific, 2010.

[9] A. Alhazov, A. Krassovitskiy, Y. Rogozhin, and S. Verlan. P systems with minimal insertion and deletion. *Theor. Comput. Sci.*, 412:136–144, 2011.

[10] A. Alhazov, C. Martín-Vide, B. Truthe, J. Dassow, and Y. Rogozhin. On networks of evolutionary processors with nodes of two types. *Fundam. Inform.*, 91(1):1–15, 2009.

[11] G. Bel, P. Dömösi, and A. Krassovitskiy. Parsing by simple insertion systems. In *Computing Languages with Multi-Agent Systems and Bio-Inspired Devices - LAMAS 2010 , Special Session of ICAART Valencia*, 2010.

[12] R. Benne. *RNA Editing: The Alteration of Protein Coding Sequences of RNA*. Ellis Horwood, Chichester, West Sussex, 1993.

[13] F. Biegler, M. J. Burrell, and M. Daley. Regulated RNA rewriting: Modelling RNA editing with guided insertion. *Theor. Comput. Sci.*, 387(2):103 – 112, 2007. Descriptional Complexity of Formal Systems.

[14] M. J. Burrell. *Computational Modelling of Uridine Insertion and Deletion in Kinetoplasid RNA*. PhD thesis, The University of Western Ontario, 2005.

[15] J. Castellanos, C. Martín-Vide, V. Mitrana, and J. M. Sempere. Solving NP-complete problems with networks of evolutionary processors. In J. Mira and A. Prieto, editors, *Connectionist Models of Neurons, Learning Processes and Artificial Intelligence, 6th International Work-Conference on Artificial and Natural Neural Networks, IWANN 2001 Granada, Spain, June 13-15, 2001, Proceedings, Part I*, volume 2084 of *Lecture Notes in Computer Science*, pages 621–628. Springer, 2001.

[16] E. Csuhaj-Varjú and J. Dassow. On cooperating/distributed grammar systems. *Elektronische Informationsverarbeitung und Kybernetik*, 26(1/2):49–63, 1990.

[17] E. Csuhaj-Varjú, C. Martín-Vide, and V. Mitrana. Hybrid networks of evolutionary processors are computationally complete. *Acta Inf.*, 41(4-5):257–272, 2005.

[18] M. Daley, L. Kari, G. Gloor, and R. Siromoney. Circular contextual insertions/deletions with applications to biomolecular computation. In *SPIRE/CRIWG*, pages 47–54, 1999.

[19] M. Domaratzki. Semantic shuffle on and deletion along trajectories. In C. Calude, E. Calude, and M. J. Dinneen, editors, *Developments in Language Theory*, volume 3340 of *Lecture Notes in Computer Science*, pages 163–174. Springer, 2004.

[20] M. Domaratzki and A. Okhotin. Representing recursively enumerable languages by iterated deletion. *Theor. Comput. Sci.*, 314(3):451–457, 2004.

[21] S. Eilenberg. *Automata, Languages, and Machines.* Academic Press, Inc., Orlando, FL, USA, 1974.

[22] R. Freund, O. Ibarra, G. Păun, and H.-C. Yen. Matrix languages, register machines, vector addition systems. In *Third Brainstorming Week on Membrane Computing*, pages 155–168. Sevilla, 2005.

[23] R. Freund, M. Kogler, Y. Rogozhin, and S. Verlan. Graph-controlled insertion-deletion systems. In I. McQuillan and G. Pighizzini, editors, *Proceedings Twelfth Annual Workshop on Descriptional Complexity of Formal Systems, Saskatoon, Canada, 8-10th August 2010*, EPTCS 31, pages 88–98, 2010.

[24] B. Galiukschov. Semicontextual grammars. *Matem. Logica i Matem. Lingvistika*, pages 38–50, 1981. Tallin University, (in Russian).

[25] J. Goldstine, M. Kappes, C. M. R. Kintala, H. Leung, A. Malcher, and D. Wotschke. Descriptional complexity of machines with limited resources. *J. Universal Computer Sci.*, 8(2):193–234, 2002.

[26] D. Haussler. *Insertion and Iterated Insertion as Operations on Formal Languages*. PhD thesis, University of Colorado at Boulder, 1982.

[27] D. Haussler. Insertion languages. *Information Sciences*, 31(1):77–89, 1983.

[28] J. Hopcroft, R. Motwani, and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Mass., 2nd edition, 2001.

[29] M. Ito, L. Kari, and G. Thierrin. Insertion and deletion closure of languages. *Theor. Comput. Sci.*, 183(1):3–19, 1997.

[30] M. Ito and R. Sugiura. n-insertion on languages. In N. Jonoska, G. Păun, and G. Rozenberg, editors, *Aspects of Molecular Computing*, volume 2950 of *Lecture Notes in Computer Science*, pages 213–218. Springer, 2004.

[31] L. Kari. *On Insertion and Deletion in Formal Languages*. PhD thesis, University of Turku, 1991.

[32] L. Kari. From micro-soft to bio-soft: Computing with DNA. In D. Lundh, B. Olsson, and A. Narayanan, editors, *Biocomputing and emergent computation: Proceedings of BCEC97*, pages 146–164. World Scientific, 1997.

[33] L. Kari, G. Păun, G. Thierrin, and S. Yu. At the crossroads of DNA computing and formal languages: Characterizing RE using insertion-deletion systems. In *Proceedings of 3rd DIMACS Workshop on DNA Based Computing*, pages 318–333. Philadelphia, 1997.

[34] L. Kari and P. Sosík. Aspects of shuffle and deletion on trajectories. *Theoretical Computer Science*, 332(1-3):47 – 61, 2005.

[35] L. Kari and P. Sosík. On the weight of universal insertion grammars. *Theor. Comput. Sci.*, 396(1-3):264–270, 2008.

[36] L. Kari and G. Thierrin. Contextual insertions/deletions and computability. *Inf. Comput.*, 131(1):47–61, 1996.

[37] I. Katsányi. A note on restricted insertion-deletion systems. *Bulletin of the EATCS*, 83:181–185, 2004.

[38] S. C. Kleene. Representation of events in nerve nets and finite automata. In C. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3–41. Princeton University Press, Princeton, NJ, 1956.

[39] A. Krassovitskiy. On the power of insertion P systems of small size. In *BWMC09*, pages 29–44, 2009.

[40] A. Krassovitskiy. On the power of small size insertion P systems. *IJCCC*, VI(2):266–277, June 2011. in press.

[41] A. Krassovitskiy, Y. Rogozhin, and S. Verlan. Computational power of P systems with small size insertion and deletion rules. In T. Neary, D. Woods, A. K. Seda, and N. Murphy, editors, *Complexity of Simple Programs 2008, CSP 2008, Cork, Ireland, December 6-7, 2008. Proceedings*, pages 137–148. Cork University Press, 2008. Invited Paper.

[42] A. Krassovitskiy, Y. Rogozhin, and S. Verlan. Further results on insertion-deletion systems with one-sided contexts. In C. Martín-Vide, F. Otto, and H. Fernau, editors, *Language and Automata Theory and Applications, Second International Conference, LATA 2008, Tarragona, Spain, March 13-19, 2008. Revised Papers*, volume 5196 of *Lecture Notes in Computer Science*, pages 333–344. Springer, 2008.

[43] A. Krassovitskiy, Y. Rogozhin, and S. Verlan. One-sided insertion and deletion: Traditional and P systems case. In E. Csuhaj-Varjú, R. Freund, M. Oswald, and K. Salomaa, editors, *International Workshop on Computing with Biomolecules, August 27th, 2008, Wien, Austria*, pages 53–64. Druckerei Riegelnik, 2008.

[44] A. Krassovitskiy, Y. Rogozhin, and S. Verlan. Computational power of insertion-deletion (P) systems with rules of size two. *Natural Computing*, 2010. In press: DOI 10.1007/s11047-010-9208-y.

[45] S. N. Krishna. *Languages of P Systems: Computability and Complexity.* PhD thesis, Indian Institute of Technology Madras, 2001.

[46] S. N. Krishna and R. Rama. Insertion-deletion P systems. In N. Jonoska and N. C. Seeman, editors, *DNA*, volume 2340 of *Lecture Notes in Computer Science*, pages 360–370. Springer, 2001.

[47] S. Marcus. Contextual grammars. *Rev. Roum. Math. Pures Appl.*, 14:1525–1534, 1969.

[48] S. Marcus, G. Paun, and C. Martín-Vide. Contextual grammars as generative models of natural languages. *Computational Linguistics*, 24(2):245–274, 1998.

[49] M. Margenstern, G. Păun, Y. Rogozhin, and S. Verlan. Context-free insertion-deletion systems. *Theor. Comput. Sci.*, 330(2):339–348, 2005.

[50] C. Martín-Vide, G. Paun, and A. Salomaa. Characterizations of recursively enumerable languages by means of insertion grammars. *Theor. Comput. Sci.*, 205(1-2):195–205, 1998.

[51] A. Matveevici, Y. Rogozhin, and S. Verlan. Insertion-deletion systems with one-sided contexts. In J. O. Durand-Lose and M. Margenstern, editors, *Machines, Computations, and Universality, 5th International Conference, MCU 2007, Orléans, France, September 10-13, 2007, Proceedings*, volume 4664 of *Lecture Notes in Computer Science*, pages 205–217. Springer, 2007.

[52] M. Minsky. *Computation: Finite and Infinite Machines.* Prentice Hall, NJ, Englewood Cliffs, 1967.

[53] V. Mitrana and B. Truthe. On accepting networks of evolutionary processors with at most two types of nodes. In A. H. Dediu, A.-M. Ionescu, and C. Martín-Vide, editors, *Language and Automata Theory and Applications, Third International Conference, LATA 2009, Tarragona, Spain, April 2-8, 2009. Proceedings*, volume 5457 of *Lecture Notes in Computer Science*, pages 588–600. Springer, 2009.

[54] M. Mutyam, K. Krithivasan, and A. S. Reddy. On characterizing recursively enumerable languages by insertion grammars. *Fundam. Inform.*, 64(1-4):317–324, 2005.

[55] K. Onodera. A note on homomorphic representation of recursively enumerable languages with insertion grammars. *Transactions of Information Processing Society of Japan*, 44(5):1424–1427, 2003.

[56] K. Onodera. New morphic characterizations of languages in Chomsky hierarchy using insertion and locality. In A. H. Dediu, A.-M. Ionescu, and C. Martín-Vide, editors, *Language and Automata Theory and Applications, Third International Conference, LATA 2009, Tarragona, Spain, April 2-8, 2009.*, volume 5457 of *Lecture Notes in Computer Science*, pages 648–659. Springer, 2009.

[57] G. Paun, G. Rozenberg, and A. Salomaa, editors. *The Oxford Handbook of Membrane Computing.* Oxford University Press, 2010.

[58] G. Păun. *Marcus Contextual Grammars.* Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[59] G. Păun. Computing with membranes. *J. Comput. Syst. Sci.*, 61(1):108–143, 2000.

[60] G. Păun. *Membrane Computing. An Introduction.* Springer-Verlag, 2002.

[61] G. Păun, M. J. Pérez-Jiménez, and T. Yokomori. Representations and characterizations of languages in Chomsky hierarchy by means of insertion-deletion systems. *Int. J. Found. Comput. Sci.*, 19(4):859–871, 2008.

[62] G. Păun, G. Rozenberg, and A. Salomaa. *DNA Computing: New Computing Paradigms.* Springer, 1998.

[63] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages.* Springer-Verlag, Berlin, 1997.

[64] W. D. Smith. DNA computers in vitro and in vivo. In R. Lipton and E. Baum, editors, *Proceedings of DIMACS Workshop on DNA Based Computers*, DI-

MACS Series in Discrete Math. and Theoretical Computer Science, pages 121–185. Amer. Math. Society, 1996.

[65] A. Takahara and T. Yokomori. On the computational power of insertion-deletion systems. In M. Hagiya and A. Ohuchi, editors, *DNA Computing, 8th International Workshop on DNA Based Computers, DNA8, Sapporo, Japan, June 10-13, 2002, Revised Papers*, volume 2568 of *Lecture Notes in Computer Science*, pages 269–280, 2002.

[66] A. Takahara and T. Yokomori. On the computational power of insertion-deletion systems. *Natural Computing*, 2(4):321–336, 2003.

[67] S. Verlan. On minimal context-free insertion-deletion systems. *Journal of Automata, Languages and Combinatorics*, 12(1-2):317–328, 2007.

[68] D. Wood. *Theory of Computation*. Harper and Row, 1987.

[69] T. Yokomori. YAC: Yet another computation model of self-assembly. IMACS Series in Discrete Mathematics and Theoretical Computer Science, 1999.

[70] M. Yong, J. Xiao-gang, S. Xian-chuang, and P. Bo. Minimizing of the only-insertion insdel systems. *J. of Zhejiang University*, 6(10):1021–1025, 2005.

[71] H. Zantema. Complexity of guided insertion-deletion in rna-editing. In A. H. Dediu, H. Fernau, and C. Martín-Vide, editors, *Language and Automata Theory and Applications, 4th International Conference, LATA 2010, Trier, Germany, May 24-28, 2010. Proceedings*, volume 6031 of *Lecture Notes in Computer Science*. Springer, 2010.