VICTOR MITRANA

NEW DEVELOPMENTS IN FORMAL LANGUAGE THEORY INSPIRED FROM BIOLOGY

Editura Universității din Bucureşti

VICTOR MITRANA

NEW DEVELOPMENTS IN FORMAL LANGUAGE THEORY INSPIRED FROM BIOLOGY

Editura Universității din București
– 2001 –

563/01

Referenţi ştiinţifici: **Cercetător pr. gr. I Gheorghe PĂUN**
Conf. dr. Alexandru MATEESCU

© Editura Universității din București
Șos. Panduri 90-92, București - 76235; Tel./Fax: 410.23.84
E-mail: editura@unibuc.ro
Internet: www.editura.unibuc.ro

Descrierea CIP a Bibliotecii Naţionale
MITRANA, VICTOR
New developments in formal language theory inspired from biology / Victor Mitrana - Bucureşti, Editura Universității din Bucureşti, 2001
p.; 21cm
Bibliogr.
ISBN: 973-575-535-1

519.765

Contents

1 Introduction **5**

1.1 The Basic Inspirations 6

1.1.1 Chomsky's Initiative in Linguistics 8

1.1.2 Inspirations from the Filamental Growth . . . 10

1.1.3 Language-Theoretic Models of Molecular Computing 12

1.1.4 Language-Theoretic Models of Genome Evolution 14

1.1.5 The Artificial Life Challenge 16

1.2 Basic Definitions 20

1.2.1 Formal Language Prerequisites 20

1.2.2 Closure Properties 25

1.2.3 Decidability 27

1.2.4 A Structural Language of Nucleic Acids 28

2 Genome Evolution: Operations **31**

2.1 Inversions, Transpositions, Duplications 31

2.1.1 Relationships Between the Above Operations . 32

2.2 A Generalization 41

2.2.1 Inclusions 42

2.2.2 Strictness of Some Inclusions 52

2.3 The Duplication Root 63

2.4 Multiple Crossing-over 67

2.5 Two Crossover Distances 79

2.5.1 Singleton Target Sets 83

2.5.2 Arbitrary Target Sets 88

3 Language Generating Devices **93**
3.1 Evolutionary Grammars 94
3.1.1 Decision Problems 97
3.1.2 A Growth Function for Genomes 102
3.2 Context-Free Evolutionary Grammars 104
3.2.1 Computational Power 106
3.2.2 Decidability Properties 113
3.2.3 Some Closure Properties 116
3.2.4 Evolutionary Grammars and the Structural Language of Nucleic Acids 118
3.2.5 Descriptional Complexity 120
3.2.6 The Differentiation Function 123
3.2.7 Adult Languages 129
3.3 Duplication Grammars 133
3.3.1 A Short Comparison 136
3.3.2 Observations on the Generative Power 139
3.3.3 Decision Problems 144
3.4 Self Crossover Systems 147

4 Other Operations **157**
4.1 The PA-Matching Operation 157
4.1.1 The Non-Iterated Case 158
4.1.2 The Iterated Case 163
4.2 The Overlapping Operation 167
4.3 Gene Assembly in Ciliates 173
4.3.1 The *ld* Operation 174
4.3.2 The *hi* Operation 183
4.3.3 The *dlad* Operation 186
4.4 Evolutionary Systems 190
4.4.1 Language of Species 195
4.4.2 Population of Evolutionary Systems 199
4.4.3 Some Growth Relation Considerations 202

Chapter 1

Introduction

The 20th century contributed to the development of human civilization in two main directions, namely the scientific understanding of universe and of human being as its part, and shifting technical, engineered devices into the interplanetary space, into the human body, and into the processes reserved before for the activities of human minds only. Cross-fertilization of different branches of science and technology has been running during the last hundred years very rapidly and it appeared very productive in many directions. The most interesting for our further intentions are the mutual influences of biology and the mathematical study of computing and natural language.

Symbols like letters in alphabets, words in languages, symbols used in chemistry, pictograms which orient us in railway-stations or airports, and many other symbols play an important role in our everyday life. Generally speaking, we live in a world of symbols similarly as we live in the real physical world. A very important feature of our intellect is that it can recognize and manipulate symbols in very different contexts. The patterns of ink in a sheet of paper we consider as letters which refer to some sounds produced by us, for instance. The sequences of such letters - words - refer to all of real or imagined things, events, or states in our real or virtual world which we are able to communicate. Complicated chemical structures can be usefully condensed into the form of short and clear structures of symbols.

All human problem solving capability can be considered in a certain sense as a manipulation with symbols and structures composed with them [99], [98].

In a rather general sense, symbols are patterns which are relatively stable in space-time with the ability to designate things other than themselves only. There are many possibilities to "materialize" symbols (ink-lines in the paper sheet, structures in the memory of a computer, etc.) but all "materializations" must have some basic properties like (relative) stability in space-time, possibility to be a part of more complicated structures - symbol structures - to be "created" and "discharged" if necessary, etc. Symbols can be organized into symbol structures - letters into words, words into sentences, pictograms into maps leading us from the entrance of an airport through the check-in desk further to the boarding-gate, the symbols A, C, G, and T, denoting the basic nucleotides, into DNA sequences which encode the genetic information of organisms, etc. We are able to designate things by using symbols or symbol structures, which represent an important attribute of symbols.

We can also recognize such structures, this is another very important attribute, and to process them in certain ways. There are different theoretically investigated and well understood possibilities of how to organize symbols into more complicated structures. The simplest structures, in a certain sense, are the strings of symbols. The formal definition of a string of symbols from a finite set of symbols, called alphabet, can be given. Also other types of structures can be defined in a formal and correct way (graphs, pictures, etc.). In a more general setting, we always are confronted with mappings between symbol systems and other systems, which can also be symbol systems.

1.1 The Basic Inspirations (Linguistics, DNA, and Computing)

Language - in its two basic forms: natural and artificial - is a particular case of symbol system. More and more approaches in science corresponds to a general process of transferring methods and tools from mathematical linguistics to other areas of research. An early and deep investigation of this phenomenon can be found in [87].

In following sections, we are going to discuss about the formal language paradigm as a possible basic link between DNA, as a part of the genetic language, computation and artificial systems. More specifically, we present some developments in the framework of formal language theory suggested by genetics and molecular biology as well as artificial life. First, we start with some considerations about the origins of the aforementioned sciences which have influenced the formal language theory.

A very significant scientific event of the 20th century was the discovery of the structure of deoxyribonucleic acid (DNA) by James D. Watson and Francis Crick in the first half of the century; a good easy-to read presentation of the story of this discovery is described in [131]. The huge macromolecules of DNA were found to be double stranded strings composed of only four types of basic nucleotides called *adenine*, *cytosine*, *guanine*, and *thymine*, abbreviated usually by the letters A, C, G and T, respectively.

Genetics is concerned in the study of the biological information of a living organism, the heredity material of all species. This information turned out to be stored in macromolecule of nucleic acids called genes and in their composition in chromosomes. Molecular biology is concerned in the relationships between nucleic acids and amino acids, between genes and proteins known to be the molecules which are responsible for almost all the functions of a living organism.

One among the most important technical achievements of the 20th century was the construction of the first electronic computers at the end of 40s. These computers were huge, of the size of a large room. The processors were based on vacuum tubes and were very expensive. So, the only way of using these machines was to execute basic computations step-by-step on a single processor. The idea of sequential computing using one processor had been supported also by prevailing theoretical results on computing theory reported during thirties and forties by Alan Turing, Emil Post, Kurt Gödel, A. Church. Despite that the notion of an algorithm has been used since Euclid and Archimedes, this notion was not made mathematically rigorous and nobody knew whether this would ever be done.

Turing imagined a device, called later *Turing machine*, which at every moment is in a state, from a finite set of states, and can scan a a cell on a arbitrarily long tape. Depending on the current state and the scanned symbol, it writes a symbol on the cell scanned, moves to the next cell to the right or to the left, and enters a new, possibly the same, state. It is easy to note that an algorithm in this approach is a sequence of symbol manipulations in a deterministic way.

1.1.1 Chomsky's Initiative in Linguistics (The Origin of Formal Grammars and Languages)

The study of symbols and symbol structures has been revolutionized by the pioneering work of Noam Chomsky. Chomsky's contribution to the study of natural languages and his invention of the notion of generative grammars ,[16], [17], the precise mathematical formalization of notions like language and grammar, [18], and the discovery of relations between formal languages, grammars and the (mathematically described) models of computing engines (like the Turing machine and numerous variants of automata) were reflected very early [63]. The way in which Chomsky's ideas influenced computer science and engineering in the sixties and seventies illustrated very impressively the power of the basic paradigm of early cybernetics.

Chomsky's view on natural languages opened a new way to consider

– words and more complicated parts of phrases as abstract symbols,

– sentences of languages as strings of symbols,

– modes of forming new strings from other oness through concatenation, and

– rules for replacement or rewriting symbols or strings by other symbols or strings.

Programming languages - artificially engineered tools for communication with computers - were designed using ideas previously considered in the study of human (natural) languages, formulated with mathematical precision, and elaborated using tools and techniques of mathematics and skills of computer programmers. It happened that the theory of formal grammars and languages loosed its original roots in traditional linguistics and raised up as a branch of theoretical computer science - as the theory of formal grammars and languages. Starting with the book [52] through the "classics" like [117], or [56] up to [114] and [126], for instance, the theory of formal grammars and languages became the traditional core of the theoretical computer science.

A good example is the use of formal grammar and language theory oriented techniques in pattern recognition [46], e.g. in recognition of different types of chromosomes [80], [81]; more on the importance of chromosomes forms from biological point of view see in [27]. In this approach, patterns are viewed as strings of symbols and the recognition process consists in an effective generation of an answer to the question whether or not a given string (representing a pattern) can or cannot be generated by a given grammar. In the positive case, the analyzed pattern belongs to the corresponding class of patterns, in the negative one it does not belong to it. We shall return to this topic in a forthcoming section.

Another interesting way of using grammars and languages was related to the learning process. It is also of practical interest how are human beings able to learn grammars from finite samples of the language they are using. The theory of syntactic inference represents an approach to this question amenable to be investigated with tools and techniques of the theory of formal grammars and languages [3]. It consists of presenting samples (finite or potentially infinite sequences) of sentences (words) with an additional information regarding the membership of just presented sentence (word) to a language in order to infer the grammar generating that language (generating all the words/sentences belonging to that language and no others), if possible. (See e.g. [2], [115])

All the mentioned approaches of using the basic grammatical paradigm to study different phenomena, were based on the sequential application of the grammar rules for generating or analysing (parsing) words in the corresponding languages.

1.1.2 Inspirations from the Filamental Growth (Lindenmayer Systems)

The idea of sequential rewriting used in the theory of formal languages was very productively modified for needs of describing parts of processes studied in biology. Complete parallel rewriting was initiated by the work of Aristid Lindenmayer, a biologist interested in the phenomenon of biological growth. Aristid Lindenmayer himself explained his motivation in the interview [72] as follows:

> After my Ph. D. in 1956 from the University of Michigan, Ann Arbor, I spent an academic year with Woodger in London, England, learning logic from him [133] and on an axiom system for life cycles [82]. (...) During these years, until 1968, I was also engaged in experimental work on development in Philadelphia and New York. Against this background came my first acquaintance with automata theory. This happened in the (academic) year 1962-63 which I spent in the biomathematics group at North Carolina State University, during which time I had the opportunity to have frequent discussions with the embryologist J. R. Gregg of Duke University. He had also worked previously with Woodger and had written papers on set-theoretical foundations of taxonomy and embryology. By this time he had discovered automata theory (which was initiated in 1956 by Moore and Mealy) and together we studied the first systematic account of this theory by Ginsburg [51]. About the same time a book [7] came out which made control theory accessible to nonmathematicians (I was already using this book in teaching a course to biologists in 1963). Our training in logic with Woodger was indispensable for us, as biologists, to be able to enter automata theory at that time.
>
> The discovery of L systems was due to a problem in Ashby's book. In a chapter on finite automata he asked the question (Exercise 4/7/7): What behavior would one expect from a long chain of coupled finite automata, re-

> ceiving inputs from each other? The Kleene's paper from 1956 on nerve nets has of course given the answer concerning a constant size network of finite automata - it has as complex behavior as a single finite automaton. But I wonder how everywhere expanding arrays of finite automaton would behave. Such arrays could provide realistic simulations of growing cellular filaments such as found in algae, mosses and fungi. The ramifications of this exercise were more extensive than I expected. But in any case, L systems came to be defined in 1968 [83] as interacting, linear, growing arrays of finite automata. (...) They are basically different from cellular automata of von Neumann and tesselation systems in that they grow not only at the edge. Eventually we came to realize that we work with grammar-like constructions. This realization resulted in an intensive cooperation with G. Rozenberg, G. Herman, and D. van Dalen after my move from New York to Utrecht in 1968. Instead of speaking of linear arrays of automata with states, inputs and outputs, and of transforming these arrays in discrete time steps to other arrays of possible different length, we then treated these structures as words over given alphabets (the set of states) and formulated context-free or context-sensitive productions which, when applied in parallel to the words, produced the following words, without distinguishing between terminal and non-terminal symbols.

Note that this motivation leads directly to parallelism. From the biological point of view, it cannot be expected that a growth runs sequentially, that the cells reproduce in some sequential order. More expectable is that reproduction runs in parallel; in each moment several cells can reproduce. So, the initiative of Lindenmayer started from the study of parallelism through the optics rooted in cybernetics and resulted in introducing parallel rewriting into the theory of formal grammars and languages.

In the theory of L systems, a colony of biological cells is represented by a string of symbols: one appearance of a symbol states for each individual cell, and different states of cells are represented by different symbols. Changes of the cell states are modeled by rewriting rules replacing symbols by other symbols or by several symbols (this is the case of reproduction) like in formal grammars. The parallel nature of the changes of cell states and cell division is modeled by the parallel execution of rewriting according to the rules in each place where symbols which can be rewritten appear. This special kind of rewriting appeared as theoretically highly interesting and was developed to a large and interesting theory of L systems as presented in, e.g. [111], [61], [113], [112].

Besides, L systems have been used in computer graphics for depicting imaginary "gardens of L" full of imaginary life forms [107]. By their simplicity and flexibility, L systems appear to be suitable to model different phenomena of *artificial life*.

1.1.3 Language-Theoretic Models of Molecular Computing

The idea that molecular complexes can be viewed as components of an information processing device dates back to the late 1950's when R. Feynman discussed the possibility of building "sub-microscopic" computers. Despite huge advances in computer miniaturization, the underlying von Neumann computational architecture has still remained the same together with its boundaries (speed, memory, etc.)

In the last decade many researchers have looked beyond these boundaries and investigated new media and computational models as quantum, optical and molecular-based computers.

In the last years it was observed an increasing interest of computer scientists for the structure of biological molecules and the way in which they can be manipulated in vitro in the aim of defining theoretical models of computation based on the genetical engineering tools.

The fundamental mechanism by which genetic material is merged is *recombination*. DNA sequences are recombined under the effect of enzymatic activities. In 1987, T. Head [59] introduced the *splicing* operation as a language theoretical approach of the recombinant behavior of DNA under the influence of restriction enzymes and ligases.

Roughly speaking, the main idea of the splicing operation is that two sequences are cut at specified sites, and the first substring of one sequence is pasted to the second segment of the other and vice versa.

A new type of computability model - called H systems - based on the splicing operations has been considered. Many variants of H systems have been invented and investigated (regulated H systems, distributed H systems, H systems with multisets, etc.) Under certain circumstances, the H systems are computationally complete and universal. This result suggests the possibility to consider the H systems as theoretical models of programmable universal DNA computers based on the splicing operation. Furthermore, a hybrid model involving grammars and splicing, connecting in a certain way the theory of grammar systems with the theory of the splicing operation was considered in [30]. Other operations on strings inspired from genetic engineering like *annealing, PA-match, cut and paste, etc.* have led to computational grammatical models. The monograph [104] presents the majority of achievements in this direction.

Another interesting model is the supercell system model (called also P-system) based on the cell membrane which serves as an interface between the interior and the exterior environments of a cell within a multicellular structure. Many and sound theoretical results have been reported. P systems are a class of distributed parallel computing devices of a biochemical inspiration borrowing ideas from Lindenmayer systems, grammar systems, the chemical abstract machine, multisets rewriting, etc.

However, it was L. Adleman who described in 1995 how a small instance of a computationally intractable problem known as the directed Hamiltonian Path Problem might be solved using molecular methods. The information is encoded as sequences of bases in DNA molecules, the algorithm employing a massively parallel random search in a test tube. Both enthusiastic and pessimistic views have been expressed, see, e.g. [104], [58], but this new idea has opened new directions of research both for computer scientists and biologists.

Significant efforts are being made now towards finding computations with practical importance which can be carried out in a molecular framework in a better way than using classical computers.

1.1.4 Language-Theoretic Models of Genome Evolution

Much of the current data for genomes is in the form of maps which are now becoming available and permit the study of the evolution of such organisms at the scale of genome for the first time ([21]).

On the other hand, there is an increasing trend throughout the field of computational biology toward abstracted, hierarchical views of biological sequences, which is very much in the spirit of computational linguistics. The last decades pointed out results and methodes in the field of formal language theory which might be applied to biological sequences. For instance, the structural representation of the syntactic information used by any parsing algorithm is a parse tree, which would appear to any biologist to be a resonable representation of the hierarchical construction of a typical gene.

We can fairly ask to what extent a grammar-based approach could be usefully generalized. Moreover, is this approach suitable to be used for computing? To further explore this question at a pragmatic level we need to implement the model and check its relevance.

Also, it may be supposed that the distinction between structural and functional or informational view of biological sequences corresponds to the conventional one drawn between syntax and semantics. The functional view will allow us to expand our horizons beyond the local phenomena of syntactical structure to large regions of DNA. It appears very important, in this respect, to define the semantics of DNA, which is mainly based on evolutionary selection, in such a way to reason linguistically about the processes of evolution as well as about the computational capacity.

The genomes of complex organisms are organized into chromosomes which contain genes arranged in linear order. It is rather commonly asserted that DNA and RNA structures can be described to a certain extent as words; for instance a DNA strand can be presented as a word over the alphabet of the four complementary pairs of nucleotides (A, T), (T, A), (C, G), (G, C). Thus DNA may be wieved as a language for specifying the structures and processes of life.

Treating chromosomes and genomes as languages raises the possibility to generalize and investigate the structural information contained in biological sequences. Despite of this view, biological sequences have not been investigated very vividly so far by methodes developed in the field of formal language theory. A pioneer's work has been reported in [11] where very simple genes were described by means of regular grammars, though different features of nucleic acids cannot be modelled by regular expressions (see the paragraph devoted to the structural language of nucleic acids at the end of this section).

Since then several approaches have been proposed so far, most investigations along these lines dealing with grammar formalisms, see, e.g., [19, 20, 53, 116, 121, 122]. Collado-Vides [19] has considered transformational grammars for modelling the gene regulations, Grate et al. [53] and Sakakibara et al. [116] considered stochastic context-free grammars for modelling RNA, and more recently, Searls [121, 122] has used definite clause grammars and cut grammars for investigating gene structure and expression or different forms of mutation and rearrangement.

The present work starts from the premise that genomes can be interpreted as languages, hence are amenable to be studied by means of the formal language theory. In the course of its evolution, the genome of an organism mutates by different processes. At the level of individual genes the evolution proceeds by local operations (point mutations) which substitute, insert and delete nucleotides of the DNA sequence. Evolutionary and functional relationships between genes can be captured by taking into considerations only local mutations ([120]). These operations viewed as operations on strings and languages have been considered from different points of view [121, 134] and the their references.

However, the analysis of the genomes of some viruses (Epstein-Barr and Herpes simplex viruses, see for instance [49], [71]) have revealed that the evolution of these viruses involved a number of large-scale rearrangements in one evolutionary event. On the other hand, comparing plant and animal mitochondrial DNA, the point mutation is estimated to be 100 times slower in plant than in animal, many genes are nearly identical (more than 99% of them are identical) in related species [100]. See also [49], for further discussions on this topic.

Chromosomal rearrangements include pericentric and paracentric inversions, intrachromosomal and interchromosomal transpositions, translocations, etc. For a description of these rearrangements, the reader is referred to [127]. The formal linguistic formulations of some known modes of rearrangements at a genomic level might be considered as follows:

- Inversion replaces a segment of a chromosome with its reverse DNA sequence.
- Transposition moves a segment to a new location in chromosome.
- Duplication copies a segment to a new location.
- Deletion cancels a segment of a chromosome.
- Crossover results in recombination of genes in a pair of homologous chromosomes by exchanging segments between parental chromatides. Crossover can be modelled as a process that exchanges segments at the end of two chromosomes.

1.1.5 The Artificial Life Challenge (Eco-Grammar Systems)

"Can we build a gradualist bridge from simple amoeba-like automata to highly purposive intentional systems, with identifiable goals, beliefs, and so forth?" asks Daniel Dennett contemplating about the philosophical background of the meaning of "artificial life" [41]. Stuart Wilson [132] proposed a research methodology for understanding intelligence through simulations of artificial life in progressively more challenging environments while retaining characteristics of holism, pragmatism, perception, and other phenomena that remain often underrepresented in traditional approaches of Artificial Intelligence (AI).

According to the pioneer of artificial life (AL) Christopher Langton [78], artificial life is the study of man-made systems that exhibit behaviors characteristic for natural living systems. It is concerned mainly with the formal basis of the life, and with tuning the behaviors of simple, low-level components - the behavors in Langton's terminology - upwards, constructing large aggregates of simple rule governed components which interact with one another non-linearly in the support of the global and complex dynamics so that the behavior that emerges at the global level of interactions of the behavors is essentially the same behavior exhibited by the natural living systems. For more information on motivations for, goals of, and techniques used in AL see [79] and [10]. The just mentioned kind of functionality of living beings is examined from different perspectives, through different optics, and using different conceptual frameworks. The field of AL concentrates towards understanding (and technical reconstruction) of the information-processing aspects of the life. In other words, it focuses to the phenomena which are identified as information-processing in their virtue.

The essential features of AL models are usually [78] summarized as follows:

– The models consist in population of simple components (programs or some formal specifications).

– There is no single program in the model that controls the interaction, cooperation or communication of all of the other programs.

– Each program in the model details the way in which a simple entity reacts to local situations in its environment, including encounters with other entities.

– There are no rules in the model that prescribe the global behavior of the modeled system.

– Any behavior of the modeled system at higher than the individual component level emerges from the lowest level individual behaviors of components.

Life and living systems (systems of living agents) form some structure. So, it can be taken somehow literally the first (of the eight) criteria associated with life in [44]: "Life is a pattern in space-time, rather than a specific material object". Accepting this, we have then to accept the idea that life, living organisms and systems of living organisms can be approached at a symbolic level, with emphasis on syntax of the above mentioned structure. This is nothing else than to say that (one of) the main framework for studying life at this level is mathematics in general and formal language theory in particular. This last assertion has also a convincing a posteriori justification: several syntactic models, such as von Neumann's cellular automata, Lindenmayer systems or Chomsky grammars in general, proved to be both adequate and relevant tools used in modeling various real life aspects.

We should stress in this place the substantial difference between the adequacy of a conceptual tool (understood as its capability of a model to fit the features of the modeled object) and its relevance (the possibility to obtain non-trivial insights about the object by studying the model, insights which cannot be obtained without using the model). In general, and roughly speaking, the adequacy is ensured by a good-inspired definition, but the relevance needs efforts in order to be proved, needs mathematical investigations which may last a significant period.

Formally, our approach fits very well with the main goal of AL, as stated in [78]: "the study of man-made systems that exhibits behaviors characteristic to natural living systems", by synthesizing "life-like behaviors within computers and other artificial media", putting emphasis on the "logical dynamics" of living systems, not necessarily on their actual chemical-physical functioning. (By the way, in spite of the debates seeming to push AL to the opposite direction, we think that AL should remain as much as possible focused on the "logical" or "syntactical", "symbolic" aspects of life, if it has to survive as an independent scientific area, not as a part of, say, biochemistry.)

In [24] the living system is modeled using the conceptual framework of the theory of formal languages and consists of several agents "living" (sensing and acting) together in a shared environment (similarly as in the case of grammar systems). However, the environment has its own dynamics. This is the reason, why ecosystems becomes in our mind as a good illustration and typical example. However, the same structure can be met in many other circumstances, as economic, social, even in artificial intelligence and computer science (collective robotics, distributed computer architectures, etc.). Therefore, the system proposed in the above mentioned article is called an eco-grammar (or EG in short) system. In this model we assume that both the agents and the environment develop (the environment independently of the agents, the agents in dependence on the environment), but the agents are able to sense and to make changes in environment.

An EG system consists of several agents described by strings of symbols, developing according to rules applied as in L systems and acting on the environment by pure rewriting rules applied sequentially, and of an environment described by a string of symbols and developing according to rules like in L systems, too. The rules used by each agent for development depend on the state of the environment. The rules used for acting on the environment depend on the state of the agent. Further features can be introduced, such as agent-to-agent action, birth and death of agents, etc. In such a way, a lot of real life-like features can be captured: changes of seasons, overpopulation, pollution, stagnation, cyclic development, immigration, hibernation, carnivorous animals, and so on and so forth. Some of the technical approaches can be found in [101].

Now, a few words about the overall organization of the book. In the next section we recall the basic concepts and notations used throught the book. Then, employing formal language theoretic framework, we consider the aforementioned operations as operations on strings and languages and investigate them with respect to some usual problems in formal language theory. It is worth mentioning here that these operations on languages have been considered in [121] and [134] as well. The operations investigated in the present book are generalizations of the operations studied in [121] but restrictions, by discarding the contexts, of those studied in [32]. Furthermore, the iterated versions of operations in debate are also considered.

Afterwards, we present a language generating mechanism based on the operations suggested by all the mutations mentioned above, following [34] in a more comprehensive way. Our results address to some classical problems in formal language theory, such as generative power, closure properties, decidability, descriptional complexity, etc. Nevertheless, some of these matters might also have biological significance. We mention that our model may be not satisfactory in order to describe the process of evolution because we take into consideration all genomes created by the given mutations whereas only some of them can or might support life. Two other language generating devices based on particular types of genome operations are further presented.

The last chapter is dedicated to other operations appearing in biochemistry, either in vivo or in vitro for which we apply the "classical" program in formal language theory: closure properties, computational power, decidability, etc.

1.2 Basic Definitions

1.2.1 Formal Language Prerequisites

We now recall some notation from formal language theory. This section offers to the reader the basic notions and notations from the formal language theory which will be used throught the book. For all undefined notions the reader is refered to [114].

An *alphabet* is always a finite set of *letters (symbols)*. For an alphabet V we denote by V^* the free monoid generated by V under concatenation; by ε the empty string, and by V^+ the free semigroup generated by V, i.e. $V^+ = V^* \setminus \{\varepsilon\}$. The elements of V^* are called *words (strings)*. The length of the string x is denoted by $|x|$, and $|x|_a$ delivers the number of occurrences of the letter a in x. The cardinality of a finite set A is denoted by $card(A)$.

Each subset of V^* is called *language* over V. For each word $x \in V^*$, $V = \{a_1, a_2, \ldots, a_n\}$, we define :

- The *Parikh mapping* ψ defined by

$$\psi(x) = (|x|_{a_1}, |x|_{a_2}, \ldots, |x|_{a_n}).$$

- The set of all the *prefixes* of x, denoted by $Pref(x)$.

- The set of all the *suffixes* of x, denoted by $Suf(x)$.

- The *mirror image* of x, denoted by $mi(x)$. If $x = a_1a_2\ldots a_n$, $a_i \in V$ for $1 \leq i \leq n$, then $mi(x) = a_na_{n-1}\ldots a_1$.

- The set of all *permutations* of $x \in V^*$, $Perm(x) = \{y \mid \psi(y) = \psi(x)\}$.

Furthermore, for a set of words A we write $\alpha(A) = \cup_{x \in A}\alpha(x)$ for all $\alpha \in \{Pref, Suf, mi, Perm\}$.

Let U and V be two alphabets, with each letter a from V one can associate a language, denoted by $s(a)$, over U. One gets an application $s : V \longrightarrow \mathcal{P}(U)$ which can be extended to V^* as follows:

$$s(\varepsilon) = \{\varepsilon\}, \qquad s(xy) = s(x)s(y),\ x, y \in V^*$$

This application is called *substitution.* Depending on the languages $s(a)$, one gets different types of substitutions. For instance, if the languages $s(a)$, for all a from V, are finite, s is called a finite substitution. In particular, those substitutions s for which the image of every letter is a word are called *(homo)morphisms.* If no language $s(a)$ contains the empty word, the morphism is called *non-erasing.* Every substitution may be extended to languages as

$$s(L) = \bigcup_{x \in L} \{s(x)\}.$$

If $L \subseteq V^*$, $k \geq 1$, and $h : V^* \longrightarrow U^*$ is a homomorphism such that $h(x) \neq \varepsilon$ for all the substrings x of any string in L, $|x| = k$, then we say that h is *k-restricted* on L. A homomorphism is said to be *restricted* if it is k restricted on some language, for some $k \geq 1$.

A *finite automaton* is an accepting device consisting of an input tape, a reading head able to scan the cells of the input tape from right to left. The device can be at any moment in a state from a finite set of states. Initially, a word is placed on the input tape, the reading head is positioned on the first letter of the word, and the automaton is in its initial state. A move of the automaton consists in reading the currently scanned symbol, changing the current state (the new state may be the former one) and moving the reading head to the next cell to the right. The input word is accepted if the automaton reads entirely this word and reaches a final state.
Formally, a nondeterministic finite automaton is a structure $A = (Q, V, f, q_0, F)$, where Q is a finite and non-empty set of states, V is

an alphabet, q_0 is the initial state, F is the set of final states, and f is a mapping $f : Q \times V \longrightarrow \mathcal{P}(Q)$. A configuration of the automaton A is determined by a pair formed from a state and a word over V.

The configuration (q, ax) moves to (s, x) if $s \in f(q, a)$, written in the form $(q, ax) \vdash (s, x)$. The reflexive and tranzitive closure of the relation $\vdash$ is denoted by $\vdash^*$.

The language recognized by the automaton A is

$$Rec(A) = \{x \in V^* | (q_0, x) \vdash^* (s, \varepsilon), s \in F\}$$

Any language recognized by a finite automaton is called *regular*. A finite automaton $A = (Q, V, f, q_0, F)$ is *deterministic* if $card(f(q, a)) \leq 1$ for all $q \in Q$ and $a \in V$. It is known that for each nondeterministic finite automaton one can construct a deterministic finite automaton such that both automata recognize the same language.

There exist important languages, as the set of all the words formed by brackets which match correctly, which are not regular. A correct word is that word which can be reduced to the empty word by iteratively removing adjacent pairs of brackets. The reader can easily find an argument for proving that the aforementioned language is not regular. However, these languages can be accepted by other automata, more powerful than finite automata, namely *pushdown automata*. A pushdown automaton is a finite automaton endowed with a pushdown memory, the next configuration of the automaton depends on the current state, input symbol currently scanned and the top symbol of the pushdown memory. Formally, a pushdown automaton is a structure $A = (Q, V, U, f, q_0, Z_0, F)$, where Q is a finite and nonempty set of states, V is the input alphabet, U is the pushdown memory alphabet, q_0 is the initial state, Z_0 is the initial content of the stack (pushdown) memory, F is the set of final states, and f is a mapping $f : Q \times (V \cup \{\varepsilon\}) \times U \longrightarrow \mathcal{P}_{\{}(Q \times U^*)$. A configuration of the automaton A is determined by a triple formed from a state, a word over V, and a word over U.

The configuration $(q, ax, A\alpha)$ moves to $(s, x, \beta\alpha)$ if $(s, \beta) \in f(q, a, A)$, written in the form $(q, ax, A\alpha) \vdash (s, x, \beta\alpha)$. The reflexive and tranzitive closure of the relation $\vdash$ is denoted by $\vdash^*$.

The language recognized by the automaton A is

$$Rec(A) = \{x \in V^* | (q_0, x, Z_0) \vdash^* (s, \varepsilon, \alpha), s \in F\}.$$

An even more powerful accepting device is the *linear bounded automaton* which has a tape whose length is linearly bounded with respect to the length of the input string, a head which is able to read a symbol from the input tape, write a symbol in the same cell, and move back and forth within the input tape. A string is accepted if the automaton starts with that string on its input tape and reaches a final state. The class of languages recognized by linear bounded automata is called the class of *context-sensitive* languages.

Now, we define some generative devices, called *grammars*.

A *grammar* is a four-tuple

$$G = (N, T, S, P),$$

where

- N and T are two disjoint alphabets whose symbols are called *nonterminals* and *terminals*, respectively.

  The nonterminal alphabet contains a distinguished nonterminal denoted by S, called the *axiom* of the grammar.

- P is a finite set of *production rules* written in the form $x \to y$, with $x \in (N \cup T)^* N (N \cup T)^*$ and $y \in (N \cup T)^*$.

The derivation relation is defined for two words $\alpha, \beta \in (N \cup T)^*$ by:

$$\alpha \Longrightarrow_G \beta \text{ iff } \begin{array}{l} \alpha = uxv, \beta = uyv \\ x \to y \in P. \end{array}$$

The index G will be omitted when it is self-understood. The language generated by G is

$$Gen(G) = \{x \in T^* | S \Longrightarrow_G^* x\}.$$

If each rule of a grammar contains just one nonterminal in its left-hand side, the grammar is called *context-free* while the language generated by such a grammar is called also context-free. The family of languages generated by context-free grammars is exactly the family of languages accepted by pushdown automata.

For example, the next context-free grammar (we listed the productions only, the other parameters can be infered immediately) generates the language of all the correctly bracketed words mentioned above:

$$S \rightarrow SS$$
$$S \rightarrow (S)$$
$$S \rightarrow \varepsilon$$

The next lemma is a necessary (but not sufficient) condition for a language to be context-free.

Lemma 1.2.1 (Pumping lemma) *For every context-free language L there exist two natural constants p, q, such that for any $z \in L$ with $|z| > p$, $z = uvwxy$ satisfying the following conditions:*

$$\begin{array}{ll} (i) & |uvw| \leq q \\ (ii) & |uv| > 0 \\ (iii) & uv^iwx^iy \in L, \textit{ for all } i \geq 0. \end{array}$$

By this lemma one can prove that the languages

$$\begin{array}{rcl} L_1 & = & \{a^nb^nc^n \mid n \geq 1\}, \\ L_2 & = & \{a^{2^n} \mid n \geq 0\}, \end{array}$$

are not context-free.

The languages generated by the arbitrary grammars are called *recursively enumerabile.* This class of languages is exactly the class of languages accepted by Turing machines.

The next results will be very useful in what follows:

Theorem 1.2.1 (Geffert Normal Form) *For each arbitrary grammar there exists an equivalent grammar (they generate the same language) $G = (\{S, A, B, C\}, T, S, P)$ having productions of the following forms, only*

$$\begin{array}{rcl} S & \rightarrow & x, x \in (\{S, A, B, C\} \cup T)^* \\ ABC & \rightarrow & \varepsilon. \end{array}$$

Let $G = (N, T, S, P)$ be an arbitrary grammar; for a derivation

$$D : S = w_0 \Longrightarrow w_1 \Longrightarrow \ldots \Longrightarrow w_n = y,\ y \in T^*$$

we define $WS(D, G) = max\{|w_j| \mid 1 \le j \le n\}$. For $y \in Gen(G)$ we write

$$WS(y, G) = min\{WS(D, G) \mid D \text{ is a derivation for } y \text{ in } G\}.$$

Theorem 1.2.2 (Working Space Theorem) *If $WS(x, G)$ for all $x \in Gen(G)$, then $Gen(G)$ is a context-sensitive language.*

1.2.2 Closure Properties

Let $\mathcal{C}$ and $\mathcal{Q}$ be two families of languages. We say that the operation

$$op : \mathcal{C}^n \longrightarrow \mathcal{Q}$$

is closed under *op* if, for any sequence of languages $L_1, L_2, \ldots, L_n \in \mathcal{C}$, $op(L_1, L_2, \ldots, L_n) \in \mathcal{C}$ holds.

We shall esspecially consider the following operations:

- Usual operations on sets: union, intersection, complementation.
- Concatenation $L_1L_2 = \{xy | x \in L_1, y \in L_2\}$.
- Substitutions.
- Inverse morphisms: if $h : V^* \longrightarrow U^*$ is a morphism, the the inverse morphism associated with h is $h^{-1} : U^* \longrightarrow \mathcal{P}(V^*)$ defined by $h^{-1}(x) = \{y \in V^* | h(y) = x\}$, for any $x \in U^*$. Moreover
$$h^{-1}(L) = \bigcup_{x \in L} h^{-1}(x),$$
for any language L.
- The Kleene closure L^* of a language L is define recursively as follows:
$$\begin{aligned} L^0 &= \{\varepsilon\} \\ L^{k+1} &= L \cdot L^k \\ L^* &= \bigcup_{k \ge 0} L^k \end{aligned}$$

- For two words $x, y \in V^*$, we define the shuffle operation

  $$Shuf(x,y) = \{x_1y_1x_2y_2 \dots x_py_p \mid x = x_1 \dots x_p, y = y_1 \dots y_p, \\ p \geq 1, x_i, y_i \in V^*, 1 \leq i \leq p\}.$$

  Furthermore, for two languages $L_1, L_2 \subseteq V^*$, we define

  $$Shuf(L_1, L_2) = \bigcup_{x \in L_1, y \in L_2} Shuf(x,y).$$

- The next operation is very similar to the previous one, defined for words of equal length only:

  $$SShuf(x,y) = a_1b_1a_2b_2 \dots a_pb_p, a_i, b_i \in V, 1 \leq i \leq p,$$

  where $x = a_1 \dots a_p, y = b_1 \dots b_p$. Naturally,

  $$SShuf(L_1, L_2) = \{SShuf(x,y) \mid x \in L_1, y \in L_2, |x| = |y|\}.$$

- A *generalized sequential machine*, shortly *gsm*, is a construct

  $$M = (Q, V, U, f, q_0, F),$$

  where Q, V, q_0, F have the same meaning as for finite automata, U is the output alphabet, and $f : (Q \times V) \longrightarrow \mathcal{P}_f(Q \times U^*)$. The relation $\vdash$ defined for finite automata is extended to *gsm*'s by

  $$(q, ax, y) \vdash (s, x, yz) \text{ dac'a } (s, z) \in f(q, a).$$

  For $x \in V^*$ we write

  $$T_M(x) = \begin{cases} \varepsilon, & \text{if } x = \varepsilon \\ y, & \text{if } (q_0, x, \varepsilon) \vdash^* (s, \varepsilon, y), s \in F, \end{cases}$$

  As usual, for any language L,

  $$T_M(L) = \bigcup_{x \in L} T_M(x).$$

  A family is closed under gsm mappings if it is closed under the operation T_M, for any *gsm M*.

The family of regular languages is closed under all operations from above while the family of context-free languages is not closed under intersection, complementation, $Shuf$ and $SShuf$.

The families in the Chomsky hierarchy are denoted by FIN, REG, LIN, CF, CS, RE: the families of finite, regular, linear, context-free, context sensitive and recursively enumerable languages, respectively. Moreover, we recall that a family $\mathcal{F}$ of languages is called a *trio*, if $\mathcal{F}$ is closed under ε-free homomorphisms, inverse homomorphisms and intersections with regular sets. It is well-known that any trio is closed under restricted homomorphisms, too (see [114]).

1.2.3 Decidability

There are lots of questions requiring algorithmic answers. A very important questions asks whether or not a word belongs to a language. This problem is known as the *membership problem*. A language for which this problem is algorithmically solvable is called *recursive*. The family of recursive languages is a proper subfamily of the family of recursively enumerable languages.

Other important decidabilty problems in the formal language theory are:

- Equivalence problem: Are the languages L_1 and L_2 equal?
- Finiteness problem: Is the language L finite?
- Inclusion problem: Is the language L_1 a subset of L_2?
- Emptyness problem: Is the language L empty?

The finiteness and emptyness problems are decidable for the class of context-free languages, but the equivalence problem is undecidable. This problem, as well as the inclusion problem, are decidable for regular languages.

The undecidability status of some problems is proved by reducing them to a famous combinatorial problem known to be undecidable. This is the Post Correspondence Problem (PCP for short): let V an alphabet with at least two letters, $n \geq 1$, and x, y two n-tuples of non-empty words over V. If

$$x = (x_1, x_2, \ldots, x_n),$$

$$y = (y_1, y_2, \ldots, y_n),$$

is an instance of PCP, we say that $PCP(x, y)$ has a solution if there exist $k \geq 1$ and $i_j \in \{1, 2, \ldots, n\}, j = 1, 2, \ldots, n$, such that

$$x_{i_1} x_{i_2} \ldots x_{i_k} = y_{i_1} y_{i_2} \ldots y_{i_k}.$$

The sequence $i_j \in \{1, 2, \ldots, n\}$ is called a solution. There is no algorithm for deciding whether or not PCP has any solution.

1.2.4 A Structural Language of Nucleic Acids

In this section we will establish some notations and recall some properties of nucleic acids complementarity [121]. The uniformly considered alphabet is the alphabet consisting of the four bases (nucleotides), namely adenine, cytosine, guanine, and thymine, abbreviated usually by the letters A, C, G and T, respectively,

$$V_{DNA} = \{A, C, G, T\}$$

and the homomorphism (called *complementarity*) $\overline{} : V_{DNA}^* \longrightarrow V_{DNA}^*$, defined by:

$$\overline{A} = T, \qquad \overline{C} = G, \qquad \overline{G} = C, \qquad \overline{T} = A$$

that corresponds to simple base complementarity.

For a DNA string w its opposite strand is $mi(\overline{w})$ because they are the strands of a double helix complementary oriented in opposite directions. We shall consider here some interesting features of DNA encoding secondary and recursive secondary RNA structure, respectively. Secondary structure we consider here is a simplification of the base-pairing within the same strand, namely a substring and its reverse complement, which are both found nearby on the same strand, fold back to base-pair with itself and form a steam-and-loop structure. We associate a linear string with each double helix, whenever a secondary structure is identified, as follows (the orientation customarily indicated by 5' and 3' is largely irrelevant for our purposes):

$$\begin{array}{l} 5' - x\alpha y mi(\bar{\alpha})z - 3' \\ \qquad\qquad\qquad\qquad\qquad \Longrightarrow \alpha mi(\bar{\alpha}) \\ 3' - \bar{x}\bar{\alpha}\bar{y}mi(\alpha)\bar{z} - 5' \end{array}$$

As one can see, from a "stem-and-loop" structure we keep the stem pattern and ignore the loop one. The set of all these linear strings consists of those strings $w \in V_{DNA}^*$ such that

$$w = mi(\overline{w})$$

or, equivalently

$$w = umi(\overline{u}), \text{ for some } u.$$

The above equivalence is a simple linguistic expression of the notion of *dyad symmetry.*

In a more general form, recursive secondary structures are common in RNA, hence in DNA which encode them, as shown in Figure 3.1.

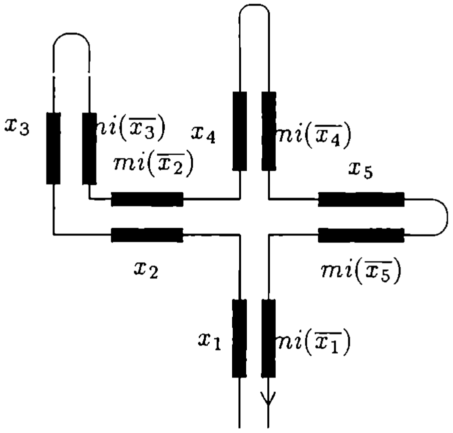

Figure 1.1.

A linear string identifying this structure can be defined as a string that leads to the empty string ε by cancelling any adjacent complementary pair $(a, \bar{a})$. These strings are called *orthodox* in [121] Denote by L_{DNA} the set of all strings defined as above. Clearly, L_{DNA} is a context-free language as shown by the context-free grammar

$$S \longrightarrow SS|aS\bar{a}|a\bar{a}$$

for all $a \in V_{DNA}$. (This is the well-known grammar for the Dyck language.)

Furthermore, we define the reduced word of a string $x \in V_{DNA}^*$ as being obtained by erasing any adjacent complementary pair from x. Obviously, the reduced string of any string is unique and the reduced string of any word in L_{DNA} is ε.

Chapter 2

Operations Suggested by the Genome Evolution

2.1 Inversions, Transpositions, Duplications

This section is dedicated to the study of some operations on strings and languages suggested by the arrangements in genomes. These operations are investigated in the frame of formal language theory; we investigate the interrelationships among them and some necessary conditions for classes of languages to be closed under these operations.

We shall not consider the crossover operation in this section because this operation, viewed as a formal operation (regardless its biological motivation and significance) is actually the splicing operation which will be investigated in more detail in a forthcoming section.

For formal language theoretic considerations with respect to deletion we refer to [70]. Some relations between inversion, transpositions and duplications very similar to those presented below are shown in [32], where lexical contexts are considered.

Let O be a pair $O = (V, O')$, where V is an alphabet and O' is a finite subset of V^+. For a string $x \in V^+$ we define the following operations with respect to the pair $O = (V, O')$:

- Inversion: $\mathcal{I}_O(x) = \{x_1 mi(x_2) x_3 \mid x = x_1 x_2 x_3,\ x_2 \in O',\ x_1, x_3 \in V^*\}$. In this case O is called inversion scheme.

- Transposition: $\mathcal{T}_O(x) = \{x_1x_3x_2x_4 \mid x = x_1x_2x_3x_4,\ x_2 \in O' \text{ or } x_3 \in O',\ x_1, x_3 \in V^*\}$. In this case O is said to be a transposition scheme.
- Duplication: $\mathcal{D}_O(x) = \{x_1x_2x_2x_3 \mid x = x_1x_2x_3,\ x_2 \in O',\ x_1, x_3 \in V^*\}$. In this case O is called duplication scheme.

If the pair O is obvious from the context, we write $\mathcal{I}$, $\mathcal{T}$, and $\mathcal{D}$ instead of $\mathcal{I}_O$, $\mathcal{T}_O$, and $\mathcal{D}_O$, respectively.

For each $S \in \{\mathcal{I}, \mathcal{T}, \mathcal{D}\}$, the operation S can naturally be extended to languages by

$$S(L) = \bigcup_{x \in L} S(x).$$

The iterated versions of the above operations are naturally defined as follows. For $S \in \{\mathcal{I}, \mathcal{T}, \mathcal{D}\}$ we set

$$\begin{aligned} S^0(L) &= L, \\ S^{i+1}(L) &= S(S^i(L)), \\ S^*(L) &= \bigcup_{i \geq 0} S^i(L). \end{aligned}$$

2.1.1 Relationships Between the Above Operations

In this section we investigate some relationships between the aforementioned operations. We shall distinguish two cases: non-iterated and iterated versions. A family $\mathcal{F}$ of languages is closed under the operation $S \in \{\mathcal{I}, \mathcal{T}, \mathcal{D}\}$, if $S_O(L) \in \mathcal{F}$ holds for all $L \in \mathcal{F}$ and any scheme O.

Non-iterated Versions

The inversion operation looks similar to the mirror image operation *mi* defined in the introductory section. It consists in the application of *mi* to a subword. However, the two operations are quite different as shown in the following proposition.

Theorem 2.1.1. *There are families of languages closed under the mirror image but not closed under inversions and vice versa.*

Proof. A D0L system is a triple $G = (V, h, w)$, where V is an alphabet, $w \in V^+$, and h is an endomorphism on V. The language generated by G is $L(G) = \{w\} \cup \{h^i(w) \mid i \geq 1\}$. It is known that the family of D0L languages is closed under mi.

Consider the D0L language

$$L = \{a^{2^n}b^{2^n} \mid n \geq 0\}$$

and the inversion scheme

$$I = (\{a, b\}, \{ab\}).$$

The language

$$\mathcal{I}_I(L) = \{a^{2^n-1}bab^{2^n-1} \mid n \geq 0\}$$

cannot be generated by a D0L system. Indeed, let us suppose that there exists a D0L system $G = (\{a, b\}, w, h)$ such that $L(G) = \mathcal{I}_I(L)$. Since $h(a^{2^n-1}bab^{2^n-1}) \in \mathcal{I}_I(L)$, for some $n \geq 2$, it follows that $|h(a)|_b = |h(b)|_a = 0$. Therefore, $h(a) = a^k$ and $h(b) = b^p$ for some $k, p \geq 1$.

If $k = p = 1$, then $L(G)$ is finite, which contradicts the infinity of $\mathcal{I}_I(L) = L(G)$.

If $k > 1$ or $p > 1$, then $h(a^{2^n-1}bab^{2^n-1})$ contains a substring of the form b^pa^k, which contradicts the form of the words in $\mathcal{I}_I(L) = L(G)$.

Now, we shall provide a family of languages closed under inversions but not closed under the mirror image. To this end, take the language

$$L_0 = \{a^nb^n \mid n \geq 1\}$$

and construct recursively the following sequence of language classes:

$$\begin{aligned}\mathcal{F}_0 &= \{L_0\},\\ \mathcal{F}_{k+1} &= \{\mathcal{I}_I(L) \mid L \in \mathcal{F}_k, I \textit{ is an inversion scheme}\}.\end{aligned}$$

The family

$$\mathcal{F} = \bigcup_{k\geq 0} \mathcal{F}_k$$

is obviously closed under inversions.

The following fact is essential in our proof.

Fact. *For every language $L \in \mathcal{F}$ and any $n \geq 1$ there exists a finite set $A(L, n) \subseteq L$ such that every string x in $L \setminus A(L, n)$ can be expressed as $x = a^p y b^q$ with $p, q \geq n$ and $y \in \{a, b\}^*$.*

If $L = L_0$, then the assertion is trivially true.

Assume that the assertion is true for any language $L' \in \mathcal{F}_k$ and take $L \in \mathcal{F}_{k+1}$. Then there exists an inversion scheme $I = (\{a, b\}, I')$ such that $L = \mathcal{I}_I(L')$. Let $n \geq 1$ be a given integer and $m = max\{|x| \mid x \in I\}$. By the induction hypothesis it follows that $L' = A(L', n + m) \cup \bar{L}$, where $A(L', n + m)$ is a finite set and every string x in $\bar{L}$ can be written as $x = a^p y b^q$, $p, q \geq n + m$. Consequently,

$$L = \mathcal{I}_I(L') = \mathcal{I}_I(A(L', n + m)) \cup \mathcal{I}_I(\bar{L}).$$

Note that $\mathcal{I}_I(A(L', n + m))$ is a finite set and any string w in $\mathcal{I}_I(\bar{L})$ can be decomposed as $w = a^r z b^s$ with $r, s \geq n$ and $z \in \{a, b\}^*$, which completes the proof of the fact.

Now it is clear that the mirror image of any language in $\mathcal{F}$ cannot be in $\mathcal{F}$ because it does not satisfy the requirements of the aforementioned fact. $\square$

We now prove that the three operations introduced above also differ in that sense that the closure under one operation does not imply the closure with respect to another one.

Theorem 2.1.2. *For any pair (X, Y) with $X, Y \in \{\mathcal{I}, \mathcal{T}, \mathcal{D}\}$, $X \neq Y$, there is a language family $\mathcal{L}$ such that $\mathcal{L}$ is closed under X and is not closed under Y.*

Proof. First we consider the family $\mathcal{F}$ defined in the second part of the proof of Theorem 2.1.1. By construction $\mathcal{F}$ is closed under inversion. On the other hand, if we apply the transposition scheme

$$T = (\{a, b\}, \{aa\})$$

to the language $L_0 \in \mathcal{F}$ we obtain a language, which contains the set of all words $a^{n-2} b^{n-1} aab$ with $n \geq 2$. This contradicts the fact shown in the proof of Theorem 2.1.1. Therefore $\mathcal{F}$ is not closed under transposition.

Let V be an alphabet. Then we consider the family $\mathcal{L}$ consisting of all languages L such that there is an integer $n \geq 1$ with $L \subseteq V^n$. Obviously, $\mathcal{L}$ is closed under inversion and transposition since these operations do not change the length of a word.

On the other hand, applying the duplication scheme

$$(V, \{a, aa\}),$$

where $a \in V$, to the language $\{a^2\} \in \mathcal{L}$ yields the language $\{a^3, a^4\}$ which is not in $\mathcal{L}$.

Let $V = \{a, b\}$. Then let $\mathcal{L}'$ be the family of all languages L over V such that each word in L can be expressed as $x_1 a x_2 b x_3$, i.e. any word of L contains ab as a scattered subword. Obviously, $\mathcal{L}'$ is closed under duplication, since duplication adds additional subwords and does not destroy scattered subwords.

On the other hand, the application of the inversion scheme (V, ab) and the transposition scheme $(V, \{a\})$ to the language $\{ab\} \in \mathcal{L}'$ yields $\{ba\} \notin \mathcal{L}'$, which proves the nonclosure of $\mathcal{L}'$ under inversion and transposition.

It remains to provide a family of languages closed under transpositions but not closed under inversions. To this end, let C be the family which contains all languages $\{a^n b^n\}$, $n \geq 1$, and is closed under transpositions. Applying the inversion scheme $(\{a, b\}, \{ab\})$ to the language $\{a^3 b^3\}$ one gets $\{a^2 bab^2\}$ which cannot be in C. □

However, the situation changes if we restrict the families of languages under consideration.

Theorem 2.1.3. *Let $\mathcal{L}$ be a family of languages which is closed under restricted homomorphisms and inverse homomorphisms. Then the following statements hold.*

i) $\mathcal{L}$ is closed under duplications if and only if it is closed under inversions.

ii) The closure of $\mathcal{L}$ under transpositions implies the closure of $\mathcal{L}$ under duplications.

iii) If $\mathcal{L}$ is closed under union or intersection with regular sets and inversions, then $\mathcal{L}$ is closed under transpositions.

Proof. i) Let L be an arbitrary language in $\mathcal{L}$ over V and let $I = (V, \{x_1, x_2, \ldots, x_n\})$ be an arbitrary inversion scheme. Then we consider $2n+1$ additional letters $c, c_1, c_2, \ldots, c_n, d_1, d_2, \ldots, d_n$ and the homomorphisms h_1, h_2, h_3, h_4 given by

$$\begin{array}{lll}
h_1 & & (V \cup \{c_1, c_2, \ldots, c_n\})^* \longrightarrow V^*, \\
 & & h_1(a) = a \text{ for } a \in V,\ h_1(c_i) = x_i,\ 1 \le i \le n, \\
h_2 & : & (V \cup \{c_1, c_2, \ldots, c_n\})^* \longrightarrow (V \cup \{c\} \cup \{c_1, c_2, \ldots, c_n\})^*, \\
 & & h_2(a) = a \text{ for } a \in V,\ h_2(c_i) = c_i c,\ 1 \le i \le n, \\
h_3 & : & (V \cup \{c_1, c_2, \ldots, c_n\} \cup \{d_1, d_2, \ldots, d_n\})^* \longrightarrow (V \cup \{c\} \cup \\
 & & \{c_1, c_2, \ldots, c_n\})^*, \\
 & & h_3(a) = a \text{ for } a \in V,\ h_3(c_i) = c_i c, h_3(d_i) = c_i cc,\ 1 \le i \le n, \\
h_4 & : & (V \cup \{c_1, c_2, \ldots, c_n\} \cup \{d_1, d_2, \ldots, d_n\})^* \longrightarrow V^*, \\
 & & h_4(a) = a \text{ for } a \in V,\ h_4(c_i) = x_i, h_4(d_i) = mi(x_i),\ 1 \le i \le n.
\end{array}$$

Then

$$\mathcal{I}_I(L) = h_4(h_3^{-1}(\mathcal{D}_D(h_2(h_1^{-1}(L))))),$$

where D is the duplication scheme which allows the duplication of the letter c only. This proves that $\mathcal{I}_I(L) \in \mathcal{L}$ holds. The converse part can be obtained in a similar way being left to the reader.

ii) Let $D = (V, \{x_i \mid 1 \le i \le n\})$ be a duplication scheme. We consider the homomorphisms

$$\begin{array}{lll}
h_1 & : & (V \cup \{c_i \mid 1 \le i \le n\})^* \longrightarrow V^*, \\
 & & h_1(a) = a \text{ for } a \in V,\ h_1(c_i) = x_i, \text{for } 1 \le i \le n, \\
h_2 & & (V \cup \{c_i \mid 1 \le i \le n\})^* \longrightarrow (V \cup \{c\} \cup \{c_1, c_2, \ldots, c_n\})^*, \\
 & & h_2(a) = a \text{ for } a \in V,\ h_2(c_i) = c_i cc, \text{for } 1 \le i \le n, \\
h_3 & : & (V \cup \bigcup_{i=1}^{n} \{c_i, d_i\})^* \longrightarrow (V \cup \{c\} \cup \{c_1, c_2, \ldots, c_n\})^*, \\
 & & h_3(a) = a \text{ for } a \in V,\ h_3(d_i) = cc_i c,\ h_3(c_i) = c_i cc,\ 1 \le i \le n, \\
h_4 & : & (V \cup \bigcup_{i=1}^{n} \{d_i, c_i\})^* \longrightarrow V^*, \\
 & & h_4(a) = a \text{ for } a \in V, h_4(d_i) = x_i x_i,\ h_4(c_i) = x_i,\ \text{for } 1 \le i \le n
\end{array}$$

and the transposition scheme

$$T = (V \cup \{c\} \cup \{c_1, c_2, \ldots, c_n\}, \{c_i \mid 1 \leq i \leq n\}).$$

Thus we get

$$\mathcal{D}_D(L) = h_4(h_3^{-1}(\mathcal{T}_T(h_2(h_1^{-1}(L))))).$$

iii) We shall give the proof in the case when $\mathcal{L}$ is closed under union. The reader can easily infer a similar construction when $\mathcal{L}$ is closed under intersection with regular sets. Obviously, if $T = (V, \{t_1, t_2, \ldots, t_n\})$ is a transposition scheme and $T_i = (V, \{t_i\})$ for $1 \leq i \leq n$, then

$$\mathcal{T}_T(L) = \mathcal{T}_{T_1}(L) \cup \mathcal{T}_{T_2}(L) \cup \cdots \cup \mathcal{T}_{T_n}(L).$$

By supposition, $\mathcal{L}$ is closed under union, and thus it is sufficient to show that $\mathcal{L}$ is closed under applications of transpositions schemes of the form $\bar{T} = (V, \{x\})$ for some $x \in V^+$. We consider the homomorphisms

$$\begin{aligned}
f_1 \ &: \ (V \cup \{c, d\})^* \longrightarrow V^*, \\
&\quad f_1(a) = a \text{ for } a \in V, \quad f_1(c) = x, \quad f_1(d) = \varepsilon, \\
f_2 \ &: \ (V \cup \{c, d\})^* \longrightarrow (V \cup \{c, d, c', d'\})^*, \\
&\quad f_2(a) = a \text{ for } a \in V, \quad f_2(c) = cc', \quad f_2(d) = dd', \\
f_3 \ &: \ (V \cup \{q, q', p, p'\})^* \longrightarrow (V \cup \{c, d, c', d'\})^*, \\
&\quad f_3(a) = a \text{ for } a \in V, \quad f_3(q) = cc', \quad f_3(q') = dd', \\
&\quad f_3(p) = c'c, \quad f_3(p') = d'd, \\
f \ & \ (V \cup \{p, p'q, q'\})^* \longrightarrow V^*, \\
&\quad f(a) = a \text{ for } a \in V, \quad f(q') = f(p) = \varepsilon, \quad f(q) = f(p') = x,
\end{aligned}$$

and the inversion schemes

$$I_1 = (V \cup \{c, d, c', d'\}, \{cc'\}) \quad \text{and} \quad I_2 = (V \cup \{c, d, c', d'\}, \{dd'\})$$

and obtain

$$\mathcal{T}_{\bar{T}}(L) = f(f_3^{-1}(\mathcal{I}_{I_1}(\mathcal{I}_{I_2}(f_2(f_1^{-1}(L)))))).$$

Note that all homomorphisms used in this proof were restricted ones.

□

Closure Properties of Some Families

We first study the closure under (non-iterated) inversion, duplication, and transposition of some language families.

Theorem 2.1.4. *Any trio is closed under duplications, transpositions and inversions.*

Proof. Let $\mathcal{F}$ be a trio. By the previous theorem it suffices to prove the closure of $\mathcal{F}$ under inversions only. We recall that all trios are closed under restricted homomorphisms [114].

Let $L \subseteq V^*$ be a language in $\mathcal{F}$ and

$$I = (V, \{x_1, x_2, \dots, x_n\})$$

be an inversion scheme. We consider the homomorphisms

$$\begin{aligned} h_1 : (V \cup \{c_i \mid 1 \leq i \leq n\})^* \longrightarrow V^*, \quad & h_1(a) = a \text{ for } a \in V, \\ & h_1(c_i) = x_i \text{ for } 1 \leq i \leq n, \\ h_2 : (V \cup \{c_i \mid 1 \leq i \leq n\})^* \longrightarrow V^*, \quad & h_2(a) = a \text{ for } a \in V, \\ & h_2(c_i) = mi(x_i) \text{ for } 1 \leq i \leq n \end{aligned}$$

and the regular set

$$R = \bigcup_{i=1}^{n} V^*\{c_i\}V^*$$

and obtain

$$\mathcal{I}_I(L) = h_2(h_1^{-1}(L) \cap R)$$

which proves the closure of $\mathcal{F}$ under inversion. □

Corollary 2.1.1. *All families in the Chomsky hierarchy are closed under duplications, transpositions and inversions.*

Iterated Versions

We now start the study of closure under iterated versions. The following lemma is a helpful tool.

Lemma 2.1.1. *Every family of languages closed under iterated inversions or iterated transpositions is closed under permutations.*

Proof. For any language $L \in V^*$ let us construct the inversion scheme

$$I = (V, \{ab \mid a, b \in V, a \neq b\})$$

and the transposition scheme

$$T = (V, V).$$

The relations

$$\mathcal{I}_I^*(L) = \mathcal{T}_T^*(L) = Perm(L)$$

follow immediately. □

Theorem 2.1.5. *The families of regular and context-free languages are closed neither under iterated inversions nor under iterated transpositions.*

Proof. Since the families of regular and context-free languages are not closed under permutations, the nonclosure with respect to iterated inversions and iterated transpositions follows by Lemma 2.1.1. □

It remains an *open problem* which of these two families are closed under iterated duplications.

Theorem 2.1.6. *The families of context-sensitive and recursively enumerable languages are closed under iterated inversions, iterated transpositions and iterated duplications.*

Proof. Let L be a context-sensitive language generated by the context-sensitive grammar $G = (N, T, S, P)$ and let $I = (T, I')$ be an inversion scheme. We construct the context-sensitive grammar $G' = (N', T, S, P')$, where

$$\begin{aligned} N' &= N \cup \{X_a \mid a \in T\}, \\ P' &= \{X_{a_1}X_{a_2}\ldots X_{a_k} \longrightarrow X_{a_k}\ldots X_{a_2}X_{a_1} \mid a_1a_2\ldots a_k \in I'\} \\ &\qquad \cup \{X_a \longrightarrow a \mid a \in T\} \\ &\qquad \cup \{h(\alpha) \longrightarrow h(\beta) \mid \alpha \longrightarrow \beta \in P,\} \end{aligned}$$

and $h : (N \cup T)^* \longrightarrow N'^*$ is the homomorphism defined by

$$h(A) = A \text{ for } A \in N \quad \text{and} \quad h(a) = X_a \text{ for } a \in T.$$

The equality $Gen(G') = \mathcal{I}_I^*(L)$ can be easily checked.

Now, we are going to prove that $\mathcal{T}_X^*(L)$ is a context-sensitive language for any transposition scheme $X = (T, \{x_i \mid 1 \leq i \leq n\})$, $n \geq 1$. To this end, we construct the phrase-structure grammar

$$\bar{G} = (\bar{N}, T, \bar{S}, \bar{P})$$

where

$$\bar{N} = N \cup \{\bar{S}, Y\} \cup \{< Y_i, y > \mid y \in Suf(x_i), 1 \leq i \leq n\}$$

and $\bar{P}$ contains – besides all rules of P – the following rules (the rules are accompanied by some informal explanations).

$$\begin{array}{rcl} \bar{S} & \longrightarrow & YS, \\ Ya & \longrightarrow & aY, \text{ and } aY \longrightarrow Ya, \ a \in T. \end{array}$$

Obvioulsly, a string of the form Yw with $w \in L$ is obtained in $\bar{G}$. The symbol Y scan the string w from left to right in order to perform nondeterministically a transposition of some x_i. If the substring x_i is identified in w, it is erased, and it will be moved to another location. This process can be done by using the following rules:

$$\begin{array}{rcl} Y & \longrightarrow & < Y_i, x_i >, \ 1 \leq i \leq n \\ < Y_i, ax > a & \longrightarrow & < Y_i, x >, \ 1 \leq i \leq n, \ a \in T \\ < Y_i, \varepsilon > a & \longrightarrow & a < Y_i, \varepsilon >, \text{ and} \\ a < Y_i, \varepsilon > & \longrightarrow & < Y_i, \varepsilon > a, \ 1 \leq i \leq n, \ a \in T \\ < Y_i, \varepsilon > & \longrightarrow & Yx_i, \ 1 \leq i \leq n. \end{array}$$

By the last set of rules, the process may be iterated. Clearly, we need also rules for finishing the process, namely $Y \longrightarrow \varepsilon$. With the above explanations we infer that $Gen(\bar{G}) = \mathcal{T}_T^*(L)$. Since the grammar $\bar{G}$

has a linear bounded working space ([114]), it follows that $\mathcal{T}_T^*(L)$ is a context-sensitive language.

By a similar proof one can show the closure under iterated duplications. The closure of the recursively enumerable languages class under these operations follows immediately. □

2.2 A Generalization

In [36] the non-iterated variants of the operations presented in the previous section are investigated in a more general framework. The results presented here are taken from [36].

Let $O = (V, O')$ be a scheme in which we allow O' to be an infinite language over V. For $\mathcal{O} \in \{\mathcal{I}, \mathcal{T}, \mathcal{D}\}$ and two families of languages $\mathcal{L}$ and $\mathcal{L}'$ we define

$$\mathcal{O}(\mathcal{L}, \mathcal{L}') = \{\mathcal{O}_O(L) \mid L \in \mathcal{L} \text{ and } O = (V, O'), O' \subseteq V^*, O' \in \mathcal{L}'\}.$$

For sake of simplicity we shall write $\mathcal{O}(L_1, L_2)$ instead of $\mathcal{O}_O(L_1)$ with $O = (V, L_2)$.

Obviously, since we can only reverse, transpose and duplicate subwords of the basic language, we obtain the following statement.

Lemma 2.2.1 *For any operation $\mathcal{O} \in \{\mathcal{I}, \mathcal{T}, \mathcal{D}\}$ and any two languages L_1 and L_2, $\mathcal{O}(L_1, L_2) = \mathcal{O}(L_1, L_2 \cap sub(L_1))$.*

Lemma 2.2.2 *If L is a language over a unary alphabet and L' is an arbitrary non-empty language, then $\mathcal{I}(L, L') = \mathcal{T}(L, L') = L$.*

Proof. By Lemma 2.2.1, it is sufficient to consider reversals and transpositions of unary words which does not change the word. □

Lemma 2.2.3 *For any $\mathcal{O} \in \{\mathcal{I}, \mathcal{T}, \mathcal{D}\}$, any finite language L and any language L', $\mathcal{O}(L, L')$ is finite.*

Proof. Since an arbitrary word w has only a finite number of subwords which can be used for the operation, $\mathcal{O}(w, L')$ is finite for any language L'. Because L is finite and $\mathcal{O}(L, L') = \bigcup_{w \in L} \mathcal{O}(w, L')$, the finiteness of $\mathcal{O}(L, L')$ follows. □

The following result follows from the definition.

Lemma 2.2.4 *For any languages families $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}'_1, \mathcal{L}'_2$ with $\mathcal{L}_1 \subseteq \mathcal{L}'_1$ and $\mathcal{L}_2 \subseteq \mathcal{L}'_2$ and any operation $\mathcal{O} \in \{\mathcal{I}, \mathcal{T}, \mathcal{D}\}$, $\mathcal{O}(\mathcal{L}_1, \mathcal{L}_2) \subseteq \mathcal{O}(\mathcal{L}'_1, \mathcal{L}'_2)$.*

2.2.1 Inclusions

The aim of this section is to prove some relations of the form $\mathcal{L} \subseteq \mathcal{O}(\mathcal{L}, \mathcal{L}')$ or $\mathcal{O}(\mathcal{L}, \mathcal{L}') \subseteq \mathcal{L}'$ for some language families $\mathcal{L}$ and $\mathcal{L}'$ and some operation $\mathcal{O}$.

We start with a direct corollary of Lemma 2.2.3.

Corollary 2.2.1 *For $\mathcal{O} \in \{\mathcal{I}, \mathcal{T}, \mathcal{D}\}$ and any language family $\mathcal{L}$, $\mathcal{O}(FIN, \mathcal{L}) \subseteq FIN$.*

This result can be extended to other families if we require some conditions for $\mathcal{L}$.

Lemma 2.2.5 *For any $\mathcal{O} \in \{\mathcal{I}, \mathcal{T}, \mathcal{D}\}$ and any language families $\mathcal{L}$ and $\mathcal{L}'$ such that $\{\varepsilon\} \in \mathcal{L}'$, $\mathcal{L} \subseteq \mathcal{O}(\mathcal{L}, \mathcal{L}')$.*

Proof. Obviously, for $\mathcal{O} \in \{\mathcal{I}, \mathcal{T}, \mathcal{D}\}$ and any language L, $L = \mathcal{O}(L, \{\varepsilon\})$. Hence, $L \in \mathcal{L}$ implies $L \in \mathcal{O}(\mathcal{L}, \mathcal{L}')$. $\square$

Clearly, there can be given other conditions in order to get $\mathcal{L} \subseteq \mathcal{O}(\mathcal{L}, \mathcal{L}')$. As an example we mention that $V \in \mathcal{L}'$ ensures $\mathcal{L} \subseteq \mathcal{I}(\mathcal{L}, \mathcal{L}')$.

Lemma 2.2.6 *If $\mathcal{L}$ is closed under morphisms, inverse morphisms and intersections with regular sets, then $\mathcal{O}(\mathcal{L}, FIN) \subseteq \mathcal{L}$ for any $\mathcal{O} \in \mathcal{U}$.*

Proof. $\mathcal{O} = \mathcal{I}$. Let L over the alphabet V be an arbitrary language in $\mathcal{L}$, and let $L' = \{w_1, w_2, \ldots, w_r\}$ be an arbitrary finite language. Then we consider r additional letters $a_1, a_2, \ldots, a_r$ and the morphisms h_1 and h_2 given by

$$h_1(a) = a \text{ for } a \in V, \quad h_1(a_i) = w_i \text{ for } 1 \leq i \leq r,$$
$$h_2(a) = a \text{ for } a \in V, \quad h_2(a_i) = mi(w_i) \text{ for } 1 \leq i \leq r.$$

Then

$$\mathcal{I}(L,L') = h_2(h_1^{-1}(L) \cap \bigcup_{i=1}^{r} V^* a_i V^*)$$

which proves that $\mathcal{I}(L,L') \in \mathcal{L}$ holds.

$\mathcal{O} = \mathcal{T}$. Let L, L' and $a_1, a_2, \ldots, a_r$ as in the preceeding considerations. Further, let $b_1, b_2, \ldots, b_r$ be additional letters. Then we define the morphisms h_1 and h_2 by

$$\begin{aligned}
&h_1(a) = a \text{ for } a \in V, \quad h_1(a_i) = w_i \text{ for } 1 \leq i \leq r,\\
&h_1(b_i) = \varepsilon \text{ for } 1 \leq i \leq r,\\
&h_2(a) = a \text{ for } a \in V, \quad h_2(a_i) = \varepsilon \text{ for } 1 \leq i \leq r,\\
&h_2(b_i) = w_i \text{ for } 1 \leq i \leq r.
\end{aligned}$$

Then

$$\mathcal{I}(L,L') = h_2(h_1^{-1}(L) \cap \bigcup_{i=1}^{r} (V^* a_i V^* b_i V^* \cup V^* b_i V^* a_i V^*))$$

which proves $\mathcal{T}(L,L') \in \mathcal{L}$.

An analogous proof can be given for $\mathcal{O} = \mathcal{D}$. □

Lemma 2.2.7 *For $\mathcal{O} \in \{\mathcal{I}, \mathcal{T}\}$and a language family $\mathcal{L} \in \{REG, CF\}$, $\mathcal{O}(REG, \mathcal{L}) \subseteq \mathcal{L}$.*

Proof. We give the proof only for $\mathcal{L} = CF$, $\mathcal{O} = \mathcal{I}$ and $\mathcal{L} = REG$, $\mathcal{O} = \mathcal{T}$. The necessary (small) modifications for the other cases can be seen from the given proofs and are left to the reader.

$\mathcal{L} = CF$, $\mathcal{O} = \mathcal{I}$. Let L and L' be a regular and context-free language, respectively. Let L be accepted by the deterministic finite automaton $A = (Z, X, \delta, z_0, F)$ and $M = (Q, X, \Gamma, \delta', q_0, \gamma_0, F')$ be the pushdown automaton accepting $mi(L')$ Then we construct the pushdown automaton

$$\begin{aligned}
K \;=\; & (Z \times (Z \cup \{t\}) \times (Z \cup \{t\}) \times (Q \cup \{t'\}), X, \Gamma, \delta'', (z_0, t, t, q_0),\\
& \gamma_0, F \times \{t\} \times \{t\} \times \{t'\})
\end{aligned}$$

where t and t' are additional symbols and, for $z_1, z_2, z_3 \in Z$, $x \in X$, $\gamma \in \Gamma$, $q \in Q$ and $q_F \in F'$, δ'' is defined as follows:

$$\delta''((z_1,t,t,q_0),x,\gamma_0) = \{((z_1',t,t,q_0),\gamma_0) \mid z_1' \in \delta(z_1,x)\}$$

(in this first phase characterized by the presence of t and the absence of t', in the first component of the state of K we simulate the work of the finite automaton A),

$$\delta''((z_1,t,t,q_0),\varepsilon,\gamma_0) = \{((z_1,z,z,q_0),\gamma_0) \mid z \in Z\}$$

(we guess that the word w under consideration under consideration can be written as $w = w_1 v w_2$ with $mi(v) \in L'$ where w_1 is the subword we have already consumed; we switch to the second phase characterized by absence of t and t'),

$$\begin{array}{rcl} \delta''((z_1,z_2,z_3,q),x,\gamma) & = & \{((z_1,z_2,z_3',q'),\gamma') \mid z_3 \in \delta(z_3',x), \\ & & (q',\gamma') \in \delta'(q,x,\gamma)\}, \\ \delta''((z_1,z_2,z_3,q),\varepsilon,\gamma) & = & \{((z_1,z_2,z_3,q'),\gamma') \mid (q',\gamma') \in \delta'(q,x,\gamma)\}, \end{array}$$

(in the second phase K simulates the work of M in the fourth component of the state and in its stack; at the same time K simulates the work of A backwards in the third component of the state; the first component of the state is not changed, i.e. it stores $\delta(z_0,w_1)$ during this phase),

$$\delta''((z_1,z_2,z_1,q_F),\varepsilon,\gamma) = \{((z_2,t,t,t'),\gamma)\}$$

(the word v read in the second phase belongs to $mi(L')$ and $z_2 = \delta(z_1, mi(v))$ holds, i.e.

$$\delta(z_0, w_1 mi(v)) = \delta(\delta(z_0,w_1), mi(v)) = \delta(z_1, mi(v)) = z_2;$$

we switch to the third phase characterized by the presence of t and t'),

$$\delta''((z_1,t,t,t'),x,\gamma) = \{((z_1',t,t,t'),\gamma') \mid z_1' \in \delta(z_1,x)\}$$

(again, we simulate the work of A). Therefore K accepts w if and only if w can be written as $w = w_1 v w_2$ such that $v \in mi(L')$ and

$\delta(z_0, w_1 mi(v) w_2) \in F$, i.e. $w_1 mi(v) w_2 \in L$. Thus K accepts $\mathcal{I}(L, L')$.

$\mathcal{L} = REG$, $\mathcal{O} = \mathcal{T}$ Let L and L' be two regular languages which are accepted by the deterministic finite automata

$$A = (Z_A, X, \delta_A, z_A, F_A) \quad \text{and} \quad B = (Z_B, X, \delta_B, z_B, F_B),$$

respectively.

We first show that the language

$$L_1 = \{u_1 u_2 v u_3 \mid u_1 v u_2 u_3 \in L, v \in L', v \neq \varepsilon, u_2 \neq \varepsilon\}$$

is regular. In order to do this we construct the nondeterministic finite automaton

$$C = (Z_C, \delta_C, X, (t_1, z_A, z_A, z_A, z_A, z_B), F_C)$$

where

$$Z_C = \{t_1, t_2, t_3, t_4\} \times Z_A \times Z_A \times Z_A \times Z_A \times Z_B$$

(t_1, t_2, t_3, t_4 denote the work on u_1, u_2, v, u_3, respectively; the second component is used to simulate the work of A on u_1 and u_3 in case of presence of t_1 or t_4 and to store the state obtained after reading u_1 in the other cases; the third component guesses the state of A obtained after reading $u_1 v$ and stores it; the fourth component simulates the work of A on u_2 starting with the guessed state; the fifth component simulates the work on v starting with the state stored in the second component; the sixth component simulates the work of B on v),

$$\begin{aligned} F_C &= \{(t_3, z_1, z_2, z_3, z_2, z_4) : z_1, z_2, \in Z_A, z_3 \in F_A, z_4 \in F_B\} \\ &\quad \cup \{(t_4, z_1, z_2, z_3, z_2, z_4) : z_2, z_3, \in Z_A, z_1 \in F_A, z_4 \in F_B\} \end{aligned}$$

(words with $u_3 = \varepsilon$ are accepted by the first set, the other words by the second set), and

$$\begin{aligned} \delta_C((t_1, z_1, z_A, z_A, z_A, z_B), x) &= \{(t_1, \delta_A(z_1, x), z_A, z_A, z_A, z_B)\} \\ &\quad \cup \{(t_2, z_1, z_2, \delta_A(z_2, x), z_A, z_B)\} \end{aligned}$$

for $z_1 \in Z_A$, $x \in X$ (M simulates the work of A on a letter x of u_1 in the second component or it reads the first letter of u_2 starting with z_2 in the fourth component and stores the state z_1 obtained after reading u_1 and the start state z_2 in the second and third component, respectively),

$$\begin{aligned}\delta_C((t_2, z_1, z_2, z_3, z_A, z_B), x) &= \{(t_2, z_1, z_2, \delta_A(z_3, x), z_A, z_B)\} \\ &\cup \{(t_3, z_1, z_2, z_3, \delta_A(z_1, x), \delta_B(z_B, x))\}\end{aligned}$$

for $z_1, z_2, z_3 \in Z_A$, $x \in X$ (M simulates the work of A on a letter x of u_2 in the fourth component or it reads the first letter of v in the fifth component starting with the state stored in the second component),

$$\delta_C((t_3, z_1, z_2, z_3, z_4, z_5), x) = \{(t_3, z_1, z_2, z_3, \delta_A(z_4, x), \delta_B(z_5, x))\}$$

for $z_1, z_2, z_3, z_4 \in Z_A$, $z_5 \in Z_B$, $x \in X$, $z_2 \neq z_4$ or $q \notin F_B$ (M simulates the work on letters of v),

$$\begin{aligned}\delta_C((t_3, z_1, z_2, z_3, z_2, z_5), x) &= \{(t_3, z_1, z_2, z_3, \delta_A(z_2, x), \delta_B(z_5, x))\} \\ &\cup\{(t_4, \delta_A(z_3, x), z_2, z_3, z_2, , z_5))\}\end{aligned}$$

for $z_1, z_2, z_3 \in Z_A$, $z_5 \in F_B$, $x \in X$ (M simulates the work on v or reads the first letter of u_3),

$$\delta_C((t_4, z_1, z_2, z_3, z_2, z_5), x) = \{(t_4, \delta_A(z_1, x), z_2, z_3, z_2, z_5)\}$$

for $z_1, z_2, z_3 \in Z_A$, $z_5 \in Z_B$, $x \in X$ (M simulates the work of A on u_3).

By the given explanations we obtain the equivalence of the following statements for a word $w = u_1 u_2 v u_3$ with $u_3 \neq \varepsilon$

a) M accepts w,

b) $\delta_A(z_A, u_1) = z_1$, $\delta_A(z_2, u_2) = z_3$, $\delta_A(z_1, v) = z_2$, $\delta_A(z_3, u_3) \in F_A$ and $\delta_B(z_B, v) \in F_B$, for some states $z_1, z_2, z_3 \in Z_A$,

c) $\delta_A(z_A, u_1 v u_2 u_3) \in F_A$ and $\delta_B(z_B, v) \in F_B$,

d) $u_1 v u_2 u_3 \in L$ and $v \in L'$,

e) $w \in L_1$.

Analogously we can show that M accepts $w' = u_1 u_2 v$ if and only if $w' \in L_1$

Thus M accepts L_1 which proves the regularity of L_1.

By a similar construction we can show that

$$L_2 = \{u_1vu_2u_3 \mid u_1u_2vu_3 \in L, v \in L', v \neq \varepsilon, u_2 \neq \varepsilon\}$$

is regular, too.

Since

$$\mathcal{T}(L, L') = L \cup L_1 \cup L_2$$

and $L, L_1, L_2 \in REG$, we obtain the regularity of $\mathcal{T}(L, L')$. □

For duplications the analogon of Lemma 2.2.7 does not hold, because

$$\mathcal{D}(\$V^*\$, \$V^*\$) = \{\$w\$\$w\$ \mid w \in V^*\}$$

implies the existence of a non-context-free language in $\mathcal{D}(REG, REG)$. Therefore we present another upper bound.

First we recall the definition of simple matrix grammars and their languages. A *k-simple matrix grammar* (with regular components) is a $(k+3)$-tuple

$$G = (N_1, N_2, \ldots, N_k, T, M, S)$$

where

- $N_1, N_2, \ldots, N_k$ and T are pairwise disjoint alphabets (the sets N_i, $1 \leq i \leq k$, are the sets of nonterminals, T is the set of terminals),
- $S \notin T \cup N_1 \cup N_2 \cup \ldots \cup N_k$ (is the axiom),
- M is a finite set of productions of the following forms:
  $S \to w$ with $w \in T^*$,
  $S \to A_1A_2 \ldots A_k$ where $A_i \in N_i$ for $1 \leq i \leq k$,
  $(A_1 \to w_1B_1, A_2 \to w_2B_2, \ldots, A_k \to w_kB_k)$ with $w_i \in T^*, B_i \in N_i$ for $1 \leq i \leq k$,
  $(A_1 \to w_1, A_2 \to w_2, \ldots, A_k \to w_k)$ with $w_i \in T^*$ for $1 \leq i \leq k$.

For $x, y \in \{S\} \cup T^* \cup T^*N_1T^*N_2 \ldots T^*N_k$, we say that x directly derives y (written as $x \Longrightarrow y$) if and only if either

$$x = S \quad \text{and} \quad S \to y \in M$$

or

$$\begin{aligned}
&x = x_1 A_1 x_2 A_2 \ldots x_k A_k, \quad \text{with } x_i \in T^*, A_i \in N_i \text{ for } 1 \leq i \leq k,\\
&y = x_1 v_1 x_2 v_2 \ldots x_k v_k,\\
&(A_1 \to v_1, A_2 \to v_2, \ldots, A_k \to v_k) \in M.
\end{aligned}$$

The language $L(G)$ generated by G is defined as

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

A language L is called a *simple matrix language* if there is an integer k and a k-simple matrix grammar G such that $L = L(G)$. By SM we denote the family of simple matrix languages. We note that $\{a^n b^n \mid n \geq 1\} \in SM$.

Lemma 2.2.8 $\mathcal{D}(REG, REG) \subseteq SM$.

Proof. Let L and L' be two regular languages L and L'. Moreover, let $G = (N, T, P, S)$ and $G' = (N', T, P', S')$ be the regular grammars (with the sets N and N' of nonterminals, the set T of terminals, the sets P and P' of productions and the axioms S and S') generating L and L', respectively. Without loss of generality we can assume that all productions in P and P' are of the form $A \to xB$ or $A \to \varepsilon$ where A and B are nonterminals and x is a terminal.

We consider the 5-simple matrix grammar

$$H = (N^{(1)}, N^{(2)}, N^{(3)}, N^{(4)}, N^{(5)}, T, P_H, S_H)$$

where, for $i \in \{1, 2, 3, 5\}$,

$$N^{(i)} = \{A^{(i)} \mid A \in N\} \cup \{X^{(i)}, Y^{(i)}\}$$

and

$$N^{(4)} = \{A^{(4)} \mid A \in N'\} \cup \{X^{(4)}\}$$

are pairwise disjoint sets (with additional letters $X^{(i)}$ and $Y^{(i)}$, $1 \leq i \leq 5$), and P_H consists of the following rules

Group 1

$S_H \to X^{(1)}X^{(2)}X^{(3)}X^{(4)}X^{(5)}$,
$(X^{(1)} \to S^{(1)}, X^{(2)} \to X^{(2)}, X^{(3)} \to X^{(3)}, X^{(4)} \to X^{(4)}, X^{(5)} \to X^{(5)})$,

Group 2
$(A^{(1)} \to vB^{(1)}, X^{(2)} \to X^{(2)}, X^{(3)} \to X^{(3)}, X^{(4)} \to X^{(4)}, X^{(5)} \to X^{(5)})$ for $A \to vB \in P$,
$(A^{(1)} \to Y^{(1)}, X^{(2)} \to A^{(2)}, X^{(3)} \to X^{(3)}, X^{(4)} \to (S')^{(4)}, X^{(5)} \to X^{(5)})$ for $A \in N$,

Group 3
$(Y^{(1)} \to Y^{(1)}, A^{(2)} \to vB^{(2)}, X^{(3)} \to X^{(3)}, C^{(4)} \to vD^{(4)}, X^{(5)} \to X^{(5)})$
for $A \to vB \in P, C \to vD \in P'$,
$(Y^{(1)} \to Y^{(1)}, A^{(2)} \to Y^{(2)}, X^{(3)} \to A^{(3)}, C^{(4)} \to Y^{(4)}, X^{(5)} \to X^{(5)})$ for $C \to \varepsilon \in P'$,

Group 4
$(Y^{(1)} \to Y^{(1)}, Y^{(2)} \to Y^{(2)}, A^{(3)} \to vB^{(3)}, Y^{(4)} \to Y^{(4)}, X^{(5)} \to X^{(5)})$ for $A \to vB \in P$,
$(Y^{(1)} \to Y^{(1)}, Y^{(2)} \to Y^{(2)}, A^{(3)} \to Y^{(3)}, Y^{(4)} \to Y^{(4)}, X^{(5)} \to A^{(5)})$ for $A \in N$,

Group 5
$(Y^{(1)} \to Y^{(1)}, Y^{(2)} \to Y^{(2)}, Y^{(3)} \to Y^{(3)}, Y^{(4)} \to Y^{(4)}, A^{(5)} \to vB^{(5)})$ for $A \to vB \in P$,
$(Y^{(1)} \to Y^{(1)}, Y^{(2)} \to Y^{(2)}, Y^{(3)} \to Y^{(3)}, Y^{(4)} \to Y^{(4)}, A^{(5)} \to Y^{(5)})$ for $A \to \varepsilon \in P$,
$(Y^{(1)} \to \varepsilon, Y^{(2)} \to \varepsilon, Y^{(3)} \to \varepsilon, Y^{(4)} \to \varepsilon, Y^{(5)} \to \varepsilon)$.

The application of the two rules of Group 1 yields the derivation

$$S_H \Longrightarrow X^{(1)}X^{(2)}X^{(3)}X^{(4)}X^{(5)} \Longrightarrow S^{(1)}X^{(2)}X^{(3)}X^{(4)}X^{(5)}.$$

Any of the first given rules of phase 2 simulates a derivation step in G and such steps can be iterated, i.e. a derivation $S \Longrightarrow^* w_1F_1$ in G is simulated. Thus we obtain the derivation

$$\begin{aligned} S^{(1)}X^{(2)}X^{(3)}X^{(4)}X^{(5)} &\Longrightarrow^* w_1(F_1)^{(1)}X^{(2)}X^{(3)}X^{(4)}X^{(5)} \\ &\Longrightarrow w_1Y^{(1)}(F_1)^{(2)}X^{(3)}(S')^{(4)}X^{(5)} \end{aligned}$$

in H. Now by iterated use of the first given rules of group 3 we simulate simultaneously a derivation $F_1 \Longrightarrow^* w_2F_2$ in G and a derivation $S' \Longrightarrow^* w_2F$ in G' because the terminals in the rules for A and C coincide. Hence we get

$$w_1Y^{(1)}(F_1)^{(2)}X^{(3)}(S')^{(4)}X^{(5)} \Longrightarrow^* w_1Y^{(1)}w_2(F_2)^{(2)}(F_2)^{(3)}w_2$$
$$F^{(4)}X^{(5)} \Longrightarrow w_1Y^{(1)}w_2Y^{(2)}(F_2)^{(3)}w_2Y^{(4)}X^{(5)}.$$

By the first given rules of group 4 we simulate a derivation $F_2 \Longrightarrow^* w_3F_3$ in G and obtain

$$w_1Y^{(1)}w_2Y^{(2)}(F_2)^{(3)}w_2Y^{(4)}X^{(5)} \Longrightarrow^* w_1Y^{(1)}w_2Y^{(2)}w_3$$
$$(F_3)^{(3)}w_2Y^{(4)}X^{(5)} \Longrightarrow w_1Y^{(1)}w_2Y^{(2)}w_3Y^{(3)}w_2Y^{(4)}(F_3)^{(5)}.$$

Finally, by the first given rules of group 5 we simulate a derivation $F_3 \Longrightarrow^* w_4F_4$ in G and terminate by the other rules which yields

$$w_1Y^{(1)}w_2Y^{(2)}w_3Y^{(3)}w_2Y^{(4)}w_4(F_4)^{(5)} \Longrightarrow^* w_1Y^{(1)}w_2Y^{(2)}w_3$$
$$Y^{(3)}w_2Y^{(4)}w_4Y^{(5)} \Longrightarrow w_1w_2w_3w_2w_4.$$

Therefore $w_1w_2w_3w_2w_4 \in L(H)$ if and only if there are derivations

$$S \Longrightarrow^* w_1F_1 \Longrightarrow^* w_1w_2F_2 \Longrightarrow^* w_1w_2w_3F_3 \Longrightarrow^*$$
$$w_1w_2w_3w_4F_4 \Longrightarrow w_1w_2w_3w_4$$

in G and $S' \Longrightarrow^* w_2$ in G'. Thus

$$L(H) = \{w_1w_2w_3w_2w_4 \mid w_1w_2w_3w_4 \in L, w_2 \in L'\}.$$

Analogously we can construct a 5-simple matrix grammar H' such that

$$L(H') = \{w_1w_3w_2w_3w_4 \mid w_1w_2w_3w_4 \in L, w_3 \in L'\}.$$

Moreover, $\mathcal{D}(L, L') = L(H) \cup L(H')$. The closure of SM under union implies $\mathcal{D}(L, L') \in SM$. $\square$

Lemma 2.2.9 *Let $\mathcal{O} \in \{\mathcal{I}, \mathcal{T}, \mathcal{D}\}$. Further, let $\mathcal{L}' \in \{CS, RE\}$ and let $\mathcal{L}$ be a language family with $\mathcal{L} \subseteq \mathcal{L}'$. Then $\mathcal{O}(\mathcal{L}, \mathcal{L}') \subseteq \mathcal{L}'$.*

Proof. We give the proof only for $\mathcal{L}' = CS$. The same proofs work for $\mathcal{L}' = RE$, too (one only has to omit the considerations on the space complexity). Moreover, we give an informal proof; the formal details are left to the reader.

$\mathcal{O} = \mathcal{I}$. It is sufficient to show that $\mathcal{I}(L, L') \in CS$ holds for any two context-sensitive languages L and L'.

We construct a Turing machine M accepting $\mathcal{I}(L, L')$. M works as follows on a given input word w:

1. M copies w on the first work tape, marks non-deterministically a subword v of w and writes a copy of v on the second worktape.

2. M replaces the marked word v by $mi(v)$ yielding $w_1 mi(v) w_2$ on the first worktape.

3. M checks $v \in mi(L')$ by taking the word on the second worktape as input and simulating the linear-bounded automaton accepting the context-sensitive language $mi(L')$.

4. M checks $w_1 mi(v) w_2 \in L$ by taking the word on the first worktape as input and simulating the linear-bounded automaton accepting L.

M accepts w if and only if both checks in phases 3 and 4 are successful. Thus $w = w_1 v w_2$ is accepted by M if and only if $w_1 mi(v) w_2 \in L$ and $v \in L'$. Hence w is accepted if and only if $w \in \mathcal{I}(L, L')$.

Moreover, phase 1 needs space $O(|w|+|v|)$. Phase 2 requires space $O(|v|)$. Phases 3 and 4 need $O(|v|)$ and $O(|w|)$, respectively. Because $|v| \leq |w|$, M needs space $O(|w|)$ on w. Since the languages accepted by Turing machines with linear space are context-sensitive, $\mathcal{I}(L, L')$ is context-sensitive.

$\mathcal{O} = \mathcal{T}$. We change the proof for $\mathcal{O} = \mathcal{I}$ as follows: In phase 2, M cancels the marked word v in the word w which yields w' and inserts v at some place in w' which yields w''. In phase 3, M checks $v \in L'$. In phase 4, M checks $w'' \in L$.

Then $w \in \mathcal{T}(L, L')$ holds if and only if w is accepted by M. Moreover, M only uses linear space.

$\mathcal{O} = \mathcal{D}$. In this case we copy two non-overlappings subwords v_1 and v_2 of the input word w on the second and third worktape. Then we check whether or not $v_1 = v_2$. If this is the case, then we copy w to the first worktape and cancel either v_1 or v_2 in the word on the first worktape. Finally, we check whether or not the word on the first worktape belongs to L and the word on the second worktape belongs to L'.

M only accepts if all checks give a positive answer. Again, we need only linear space which proves $\mathcal{D}(L, L') \in CS$. □

2.2.2 Strictness of Some Inclusions

The preceeding lemmas prove some inclusion of the form $\mathcal{L} \subseteq \mathcal{O}(\mathcal{L}, \mathcal{L}') \subseteq \mathcal{L}'$. The following lemmas show the properness of the inclusions in some cases. For this it is sufficient to show that there are languages in $\mathcal{L}' \setminus \mathcal{O}(\mathcal{L}, \mathcal{L}')$ and $\mathcal{O}(\mathcal{L}, \mathcal{L}') \setminus \mathcal{L}$, respectively.

Lemma 2.2.10 *Let $\mathcal{O} \in \{\mathcal{I}, \mathcal{T}, \mathcal{D}\}$. $\{a^n b^n \mid n \geq 1\} \neq \mathcal{O}(R, Q)$ for any regular language R and any language Q.*

Proof. $\mathcal{O} = \mathcal{I}$. Let $L = \{a^n b^n \mid n \geq 1\}$. Let us assume that $L = \mathcal{I}(R, Q)$ for some regular language R and some language Q. Without loss of generality we assume that there is no language $Q' \subset Q$ such that $L = \mathcal{I}(R, Q')$. This ensures that any word in Q is reversed as a subword of some word in R.

If $a^n b^n \in \mathcal{I}(z, q)$ for some words $z \in R$ and $q \in Q$, then one of the following conditions hold:

– $z = a^n b^n$ and $q \in a^+$ or
– $z = a^n b^n$ and $q \in b^+$ or
– $z = a^{n_1} b^{i_2} a^{n_3} b^{n_4}$, $q = b^{n_2} a^{n_3}$ and $n_1 + n_3 = n_2 + n_4 = n$.

Hence

$$\begin{aligned} \mathcal{I}(R, Q) \quad = \quad & \mathcal{I}(R \cap a^+ b^+, Q \cap a^+) \cup \mathcal{I}(R \cap a^+ b^+, Q \cap b^+) \cup \\ & \mathcal{I}(R \cap a^+ b^+ a^+ b^+, Q \cap b^+ a^+). \end{aligned} \tag{2.1}$$

Because the reversal of subwords in a^+ or b^+ does not change the word

$$\mathcal{I}(R \cap a^+ b^+, Q \cap a^+) = R \cap a^{l_a} a^* b^+$$

and

$$\mathcal{I}(R \cap a^+b^+, Q \cap b^+) = R \cap a^+b^{l_b}b^*$$

where l_a and l_b are the smallest lengths of words in $Q \cap a^+$ and $Q \cap b^+$, respectively. Therefore $\mathcal{I}(R \cap a^+b^+, Q \cap a^+)$ and $\mathcal{I}(R \cap a^+b^+, Q \cap b^+)$ are regular languages.

We now distinguish two cases.

Case 1: $Q \cap b^+a^+$ is finite. Then $\mathcal{I}(R \cap a^+b^+a^+b^+, Q \cap b^+a^+)$ is regular by Lemma 2.2.6. Therefore, by (2.1), $\mathcal{I}(R, Q)$ is regular. Since L is not regular, we get a contradiction to $L = \mathcal{I}(R, Q)$.

Case 2: $Q \cap b^+a^+$ is infinite. Then by the well-known fact that any set of Parikh vectors of an infinite set contains two comparable elements we obtain the existence of two different words $b^ka^{k'}$ and $b^ma^{m'}$ with $0 < k \leq m$ and $0 < k' \leq m'$. Because $a^sb^ma^{m'}b^t \in R$ where $s + m' = m + t = n$ for some n (the reversal of $b^ma^{m'}$ leads to a^nb^n), we get by the reversal of $b^ka^{k'}$ that $w = a^sb^{m-k}a^{k'}b^ka^{m'-k'}b^t$ is in $\mathcal{I}(R, Q)$. However, since $m - k > 0$ or $m' - k' > 0$ has to hold, $w \notin L$. This contradicts $L = \mathcal{I}(R, Q)$.

$\mathcal{O} = \mathcal{T}$. Let us assume that $L = \mathcal{T}(R, Q)$ for some regular language R and some language Q. Again, we assume that there is no language $Q' \subset Q$ with $L = \mathcal{T}(R, Q')$. Since L is not regular and $\mathcal{T}(R, \{\varepsilon\}) = R$, Q contains a non-empty word w. Now let $a^nb^n \in \mathcal{T}(R, w)$. We distinguish three cases.

Case 1: $w = a^m$ for some $m \geq 1$. Then $a^{n-m}b^na^m \in \mathcal{T}(R, w) \subseteq \mathcal{T}(R, Q)$. Since $a^{n-m}b^na^m \notin K$ we obtain a contradiction to $K = \mathcal{T}(R, Q)$.

Case 2: $w = a^ib^j$ for some $i \geq 1, j \geq 1$. Hence $a^{n-i}b^{n-j}a^ib^j \in \mathcal{T}(R, Q)$ which leads to a contradiction as above.

Case 3: $w = b^m$ for some $m \geq 1$. Hence $b^ma^nb^{n-m} \in \mathcal{T}(R, Q)$ which leads to a contradiction, again.

$\mathcal{O} = \mathcal{D}$. First we mention that Q cannot be finite, since $\mathcal{D}(REG, FIN) \subseteq REG$ by Lemma 2.2.6 and L is not regular. Furthermore, we can assume that Q contains only words of $a^+ \cup b^+$ because otherwise the duplication leads to words containing two times the subword ab or ba and thus not contained in L. Therefore Q contains at least a^i and a^j or at least b^i and b^j for some integers i and j with $1 \leq i < j$.

Let $a^i, a^j \in Q$ and $a^n b^n \in \mathcal{D}(R, a^j)$. Then $a^{n-2(j-i)} b^n \in \mathcal{D}(R, a^i)$. This contradicts $L = \mathcal{D}(R, Q)$. Analogously we get a contradiction if $b^i, b^j \in Q$. □

Let $h : \{a,b\}^* \to \{a,b\}^*$ be the morphism defined by

$$h(a) = ab \quad \text{and} \quad h(b) = ba \ .$$

Then we set

$$T = \bigcup_{i \geq 1} \{h^i(a)\}.$$

It is known that T is not context-free does not contain subwords of the form ww where $|w| \geq 2$ (see [15]) and that T is not context-free (use a pumping lemma).

Lemma 2.2.11 *For any context-free language L and any language Q, $T \neq \mathcal{D}(L, Q)$ holds.*

Proof. Let us assume that $T = \mathcal{D}(L, Q)$ for some context-free language L and some language Q. Since $\mathcal{D}(CF, FIN) \subseteq CF$ by Lemma 2.2.6, Q has to be infinite. Thus Q contains a word w with $|w| \geq 2$ and there is a word $v \in T$ such that $v \in \mathcal{D}(L, w)$. Obviously, $v = u_1 w u_2 w u_3$ with $u_1 w u_2 u_3 \in L$ or $u_1 u_2 w u_3 \in L$. Hence $v_1 = u_1 w w u_2 u_3 \in \mathcal{D}(L, w)$ or $v_2 = u_1 u_2 w w u_3 \in \mathcal{D}(L, w)$, and therefore $v_1 \in \mathcal{D}(L, Q)$ or $v_2 \in \mathcal{D}(L, Q)$. However, $v_1, v_2 \notin T$, which is a contradiction to $T = \mathcal{D}(L, Q)$. □

Lemma 2.2.12 *For $\mathcal{O} \in \{\mathcal{I}, \mathcal{T}\}$, $\mathcal{L} \in \mathcal{K}$ and $\mathcal{L}' \in \{CS, RE\}$ with $\mathcal{L} \subset \mathcal{L}'$, $\mathcal{O}(\mathcal{L}, \mathcal{L}') \subset \mathcal{L}'$.*

Proof. By Lemma 2.2.9, $\mathcal{O}(\mathcal{L}, \mathcal{L}') \subseteq \mathcal{L}'$. In order to show the strictness of this inclusion we consider a unary language $K \in \mathcal{L}' \setminus \mathcal{L}$ and assume that $K \in \mathcal{O}(\mathcal{L}, \mathcal{L}')$. Then $K = \mathcal{O}(L, L')$ for some languages $L \in \mathcal{L}$ and $L' \in \mathcal{L}'$. If $alph(K) = \{a\}$, then $K = \mathcal{O}(L \cap a^*, L' \cap a^*)$ also holds. Since $L \cap a^*$ is a unary language in $\mathcal{L}$, by Lemma 2.2.2 we obtain $K = L \cap a^*$. This implies $K \in \mathcal{L}$ in contradiction to its choice. □

Lemma 2.2.13 *Let $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$ be language families such that $REG \subseteq \mathcal{L}_1$, $\mathcal{L}_2 \setminus \mathcal{L}_3$ contains a non-empty language, $\mathcal{L}_2$ is closed under reversal and product with letters and $\mathcal{L}_3$ is closed under right and left quotient with letters. Then $\mathcal{I}(\mathcal{L}_1, \mathcal{L}_2) \setminus \mathcal{L}_3$ contains a non-empty language, too.*

Proof. By supposition, there is a language $L \in \mathcal{L}_2 \setminus \mathcal{L}_3$. Let $V = alph(L)$ and $\$ \notin V$. Then $\$mi(L)\$ \in \mathcal{L}_2$ by the required closure properties of $\mathcal{L}_2$. Thus $\$L\$ = \mathcal{I}(\$V^*\$, \$mi(L)\$) \in \mathcal{I}(\mathcal{L}_1, \mathcal{L}_2)$. Furthermore, by the closure of $\mathcal{L}_3$ under quotients with letters $\$L\$ \notin \mathcal{L}_3$ which proves the statement. □

Lemma 2.2.14 *Let $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$ be language families such that $REG \subseteq \mathcal{L}_1$, $\mathcal{L}_2 \setminus \mathcal{L}_3$ contains a non-empty language, $\mathcal{L}_2$ is closed under product with letters and $\mathcal{L}_3$ is closed under right and left quotient with letters. Then $\mathcal{T}(\mathcal{L}_1, \mathcal{L}_2) \setminus \mathcal{L}_3$ contains a non-empty language, too.*

Proof. By supposition, there is a language $L \in \mathcal{L}_2 \setminus \mathcal{L}_3$. Let $V = alph(L)$ and $\$ \notin V$. Then $\$L\$ \in \mathcal{L}_2$ and $\$L\$ = \mathcal{T}(\$V^*\$, \$L\$) \in \mathcal{T}(\mathcal{L}_1, \mathcal{L}_2)$. This leads to a contradiction as in the proof of Lemma 2.2.13. □

Lemma 2.2.15 *For $\mathcal{O} \in \{\mathcal{I}, \mathcal{T}, \mathcal{D}\}$ and language families $\mathcal{L}$ and $\mathcal{L}'$ of $\mathcal{K}$ such that $REG \subseteq \mathcal{L} \subset \mathcal{L}'$, $\mathcal{L} \subset \mathcal{O}(\mathcal{L}, \mathcal{L}')$.*

Proof. By Lemma 2.2.5, $\mathcal{L} \subseteq \mathcal{O}(\mathcal{L}, \mathcal{L}')$. We have to show the strictness of the inclusion.

For $\mathcal{O} \in \{\mathcal{I}, \mathcal{T}\}$, this follows from Lemmas 2.2.13 and 2.2.14 with $\mathcal{L} = \mathcal{L}_1 = \mathcal{L}_3$ and $\mathcal{L}' = \mathcal{L}_2$.

Now, let $L \in \mathcal{L}' \setminus \mathcal{L}$, $V = alph(L)$ and $\$ \notin V$. Then

$$L' = \{\$w\$\$w\$ \mid w \in L\} \in \mathcal{O}(\$L\$, \$V^*\$).$$

Let us assume that $L' \in \mathcal{L}$.

If $\mathcal{L} \in \{REG, CF, RE\}$, we can construct a generalized sequential machine M such that its induced function f_M satisfies $f_M(L') = L$. By the closure properties of the families under consideration $f_M(L') \in \mathcal{L}$ which gives $L \in \mathcal{L}$ in contrast to the supposition.

If $\mathcal{L} = CS$, we first construct the ε-free generalized sequential machine N which induced function f_N satisfies $f_N(L') = \{\$w\$^{|w|+3} \mid w \in L\}$. Again, $f_N(L') \in CS$. Then there exists a Turing machine P which accepts $f_N(L')$ with linear space. Now we construct the Turing machine P' which works as follows on the input w. It copies w to the first worktape and adds one symbol \$ before w and $|w| + 3$ symbols \$ after w on the first worktape. Then it simulates P where the first worktape is considered as the input tape. P' accepts if and only P accepts. Therefore P' accepts L. Moreover, P' needs only linear space. Thus $L \in CS$ in contrast to the supposition. $\square$

We list below a few further relations which are direct consequences of the results presented so far.

- $CF \subset \mathcal{I}(CF, REG)$. The inclusion follows from Lemma 2.2.5 and strictness from

$$\begin{aligned} K &= \{\$w\$w\$: w \in V^*\} \notin CF, \\ K &= \mathcal{I}(\{\$w\$mi(w)\$: w \in V^*\}, \{\$w\$: w \in V^*\}) \in \\ &\quad \mathcal{I}(CF, REG) \end{aligned}$$

  where V is an arbitrary alphabet with $\$ \notin V$.

- $CF \subset \mathcal{T}(CF, REG)$. Again, the inclusion follows from Lemma 2.2.5. In order to show the strictness we consider the language

$$K = \mathcal{T}(\{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\}, b^+) \in \mathcal{T}(CF, REG).$$

  If K is context-free, then $K' = K \cap a^+c^+b^+d^+$ is also context-free. However, this contradicts

$$K' = \{a^n b^m c^n d^m \mid n \geq 1, m \geq 1\} \notin CF$$

  which can be shown easily by pumping techniques. Therefore $K \notin CF$ is valid.

Finally in this section we note that, for $\mathcal{O} \in \{\mathcal{I}, \mathcal{T}\}$, the results of the form $\mathcal{O}(\mathcal{L}_1, \mathcal{L}_2) \subset \mathcal{L}_2$ with $\mathcal{L}_1 \subset \mathcal{L}_2$ are almost "optimal". This follows from Lemma 2.2.13 and Lemma 2.2.14 which say that a family

$\mathcal{L}_3$ with $\mathcal{O}(\mathcal{L}_1, \mathcal{L}_2) \subseteq \mathcal{L}_3 \subset \mathcal{L}_2$ cannot be closed under quotients with letters and the fact that most of the interesting language families are closed under quotients with letters.

We mention that in the case of duplication we do not have such "optimalities" as for inversion and transposition. Moreover, it is open whether or not the inclusions $CS \subseteq \mathcal{D}(CS, RE) \subseteq RE$ are strict.

A similar investigation remains to be done for iterated versions of these operations. Along the same lines, the following problem naturally arises. For a language $L \subseteq V^*$, we set

$$\begin{aligned} D(L) &= \{uxxv \mid uxv \in L, u, x, v \in V^*\}, \\ D^0(L) &= L, \\ D^i(L) &= D(D^{i-1}(L)), i \geq 1, \\ D^*(L) &= \bigcup_{i \geq 0} D^i(L). \end{aligned}$$

Otherwise stated, $D^*(L)$ is the smallest language $L' \subseteq V^*$ such that $L \subseteq L'$ and whenever $uxv \in L'$, $uxxv \in L'$ holds for all $u, x, v \in V^*$.

For singleton languages $L = \{w\}$ and $i \geq 0$, we write $D^i(w)$ and $D^*(w)$ instead of $D^i(\{w\})$ and $D^*(\{w\})$, respectively.

The problem asks whether $D^*(L)$ is still regular/context-free provided L is regular/context-free. If one restricts to the two-letter alphabet, then the following result holds [39].

Theorem 2.2.1 *If L is a regular language over the two-letter alphabet, then $D^*(L)$ is also regular.*

Proof. First, we prove that if w is a string over the two-letter alphabet, then $D^*(w)$ is a regular language. The assertion is trivial provided that w is the empty string or $w \in a^+$.

Let $V = \{a, b\}$ and $w = a_1 a_2 \ldots a_n$, where $a_i \in V$ for $1 \leq i \leq n$, $n \geq 2$. We prove that

$$\begin{aligned} D^*(w) = \{ & w_1 a_1 w_2 a_2 w_3 a_3 \ldots w_{n-1} a_{n-1} w_n a_n w_{n+1} \mid w_1 \in a_1^*, \\ & w_{n+1} \in a_n^*, w_i \in a_i^* \text{ for } a_{i-1} = a_i, w_i \in V^* \\ & \text{for } a_{i-1} \neq a_i, 2 \leq i \leq n\}. \end{aligned} \qquad (2.2)$$

We first prove the inclusion $\supseteq$. Obviously $w_1 \in a_1^*$ can be generated by duplications of a_1. Analogously, $w_{n+1} \in a_n^*$ can be generated by duplications of a_n. If $a_{i-1} = a_i$, then we can generated $w_i \in a_i^*$ by duplicatios of a_i.

Now let $a_{i-1} \neq a_i$. Without loss of generality we assume that $a_{i-1} = a$ and $a_i = b$. We distinguish four cases for $w_i \in \{a, b\}^*$.

Case 1. $w_i = b^{i_1}a^{j_1}b^{i_2}a^{j_2}\ldots b^{i_k}a^{j_k}$.

Then we first duplicate k-times the word $ab = a_{i-1}a_i$ which leads to the word

$$a_1a_2\ldots a_{i-1}(ab)^{k+1}a_{i+1}a_{i+2}\ldots a_n =$$
$$a_1a_2\ldots a_{i-1}a_{i-1}(ba)^k a_i a_{i+1}a_{i+2}\ldots a_n.$$

The desired powers i_l and j_l, $1 \le l \le k$, can be obtained by duplications of the corresponding letter.

Case 2. $w_i = b^{i_1}a^{j_1}b^{i_2}a^{j_2}\ldots b^{i_{k-1}}a^{j_{k-1}}b^{i_k}$.

Then we first duplicate $a_i = b$ i_k-times and proceed as in Case 1 to get $b^{i_1}a^{j_1}b^{i_2}a^{j_2}\ldots b^{i_{k-1}}a^{j_{k-1}}$.

Case 3. $w_i = a^{i_1}b^{j_1}a^{i_2}b^{j_2}\ldots a^{i_k}b^{j_k}$.

We first duplicate $a_{i-1} = a$ and $a_i = b$ i_1-times and j_k-times, respectively and proceed as in Case 1 in order to get the string $b^{j_1}a^{i_2}b^{j_2}\ldots a^{i_{k-1}}b^{j_{k-1}}a^{i_k}$.

Case 4. $w_i = a^{i_1}b^{j_1}a^{i_2}b^{j_2}\ldots a^{i_{k-1}}b^{j_{k-1}}a^{i_k}$.

Now we first duplicate $a_{i-1} = a$ i_1-times and proceed then as in Case 1 to obtain $b^{j_1}a^{i_2}b^{j_2}\ldots a^{i_{k-1}}b^{j_{k-1}}$.

The converse inclusion is proved by induction. Denote by L the language in the righthand side of equation 2.2. Obvioulsy, $D^0(w) \subseteq L$. Assume that $D^n(w)$ is included in L and take an arbitrary string x from $D^{n+1}(w)$. Assume that x is obtained from $y \in D^n(w)$ by duplicating one of its substrings. Since $y \in L$ we can write:

$$\begin{aligned} y &= w_1a_1w_2a_2w_3a_3\ldots w_{n-1}a_{n-1}w_na_nw_{n+1} \mid w_1 \in a_1^*, w_{n+1} \in a_n^*, \\ &\quad w_i \in a_i^* \text{ for } a_{i-1} = a_i, w_i \in V^* \text{ for } a_{i-1} \neq a_i, 2 \le i \le n, \\ x &= w_1a_1w_2a_2\ldots u_1u_2a_i\ldots v_1u_2a_i\ldots v_1v_2a_j\ldots a_{n-1}w_na_nw_{n+1}, \end{aligned}$$

with

$$\begin{aligned} w_i &= u_1u_2, u_1u_2 \in V^*, \\ w_j &= v_1v_2, v_1v_2 \in V^* \end{aligned}$$

for some $j > i$. In other words, the substring of y that has been duplicated was $u_2a_iw_{i+1}a_{i+1}\dots a_{j-1}v_1$. Note that the string obtained from y by duplicating a substring of some w_s is trivially in L. Therefore we consider $j > i$.

If a_{i-1} and a_i are distinct letters, then x can be written in the form

$$x = w_1a_1w_2a_2\dots w_i'a_i\dots w_ja_j\dots a_{n-1}w_na_nw_{n+1},$$

where $w_i' = w_ia_iw_{i+1}a_{i+1}\dots v_1u_2$ which is a string in $\{a,b\}^*$, hence $x \in L$.

If $a_{i-1} = a_i$, then we distinguish two cases:

Case 1. $a_i = a_{i+1} = \dots a_j$. By the definition of L, it follows that x is in L.

Case 2. $a_i = a_{i+1} = \dots a_{k-1} \neq a_k$ for some $k > i$. Then x can be written as

$$x = w_1a_1w_2a_2\dots w_ia_i\dots w_k'a_k\dots w_ja_j\dots a_{n-1}w_na_nw_{n+1},$$

where $w_k' = w_ka_k\dots v_1u_2a_iw_{i+1}\dots w_k$ which is in $\{a,b\}^*$. Hence $x \in L$ holds which completes the proof of our assertion.

Now, let M be a *gsm* which translates any string $w = a_1a_2\dots a_n \in \{a,b\}^*$ in $T_M(w) = b_1a_1b_2a_2\dots b_na_nb_{n+1}$, where

$$b_i = \begin{cases} [a_1], \text{ if } i = 1, \\ [a_n], \text{ if } i = n+1, \\ [a_i], \text{ if } a_{i-1} = a_i,\ 2 \leq i \leq n, \\ [ab], \text{ if } a_{i-1} \neq a_i,\ 2 \leq i \leq n. \end{cases}$$

Consider also the regular substitution $s : \{a, b, [a], [b], [ab]\}^* \longrightarrow 2^{\{a,b\}}$ defined by

$$s(a) = \{a\},\ s(b) = \{b\},\ s([a]) = a^*,\ s([b]) = b^*,\ s([ab]) = \{a,b\}^*.$$

As $D^*(L) = \bigcup_{w\in L} D^*(w)$ and by the proof of the assertion from above, we obtain $D^*(L) = s(T_M(L))$, which implies the regularity of $D^*(L)$. □

The problem has been completely solved for singleton languages over arbitrary alphabets by Ming in [92] in the following way.

Theorem 2.2.2 *If w is a string containing at least three different symbols, then $D^*(w)$ is not regular.*

Proof. We assume that $w = abc$ and $V = \{a, b, c\}$ below. The general case follows easily from this.

Fact 1. *Suppose that $u = abcu'$, where $u' \in V^*$; then there exists $v \in V^*$ such that $uv \in D^*(w)$.*

Proof of the fact. We show how to construct $z = uv$ iteratively. Initially, we set $z = abc$. Suppose that $u = a_1a_2\dots a_k$ and $z = b_1b_2\dots b_l$. First we have that $a_i = b_i$ for $1 \le i \le 3$. Then for each $4 \le i \le k$, we do the following: we find the largest index $j < i$ such that $b_j = a_i$. Then we duplicate the subword of z determined by the indices $j \dots i-1$. The effect of this duplication is to make the prefixes of u and z agree on all indices up to and including i. For example, suppose $u = abcbacca$; we construct z iteratively as follows, where the underlined portion shows the subword which is to be duplicated:

$$a\underline{bc} \longrightarrow \underline{abcb}c \longrightarrow ab\underline{cba}bcbc \longrightarrow abcba\underline{c}babcbc \longrightarrow$$
$$abcb\underline{acc}babcbc \longrightarrow abcbaccaccbabcbc,$$

which concludes the proof of the first fact.

Fact 2. *Let $t(x)$ be the minimal number of duplications necessary to get x from w. We have $t(x) \ge log_2(|x|/3)$.*

Proof of the fact. Each duplication at most doubles the length of the previous word and the starting word is of length 3.

Fact 3. *Suppose that $u = abcu' \in V^*$ is square-free (it does not contain repetitions). Let v be the shortest word such that $uv \in D^*(w)$. Then $|v| \ge log_2(|u|/3)$.*

Proof of the fact. By the definition of t, uv is obtained from w by a sequence of at least $t(uv)$ duplications. Since u is square-free, each of these duplications must result in at least one additional letter outside u, i.e., in v. It follows that

$$|v| \geq t(uv) \geq \log_2(|uv|/3) \geq \log_2(|u|/3)$$

and the proof of the fact is complete.

We are now ready to prove the theorem using Myhill-Nerode's characterization of regular languages. We construct an infinite sequence of pairwise inequivalent words as follows. We start by defining $W_1 = abc$. For $i \geq 1$, we define W_{i+1} inductively as follows: let V_i be such that $W_iV_i \in D^*(w)$. Then we choose W_{i+1} to be a square-free word starting with abc, such that $\log_2(|W_{i+1}|/3) > |V_i|$. Such a word exists because there are infinitely many square-free words over a three-letter alphabet. This length condition ensures (by the third fact) that $W_{i+1}V_j \notin D^*(w)$ for all $j \leq i$. It follows that W_i are pairwise inequivalent, which implies that $D^*(w)$ is not regular. □

However,

Theorem 2.2.3 *For all $w \in V^*$ and all $a_1, a_2, \ldots, a_n \in V$, the language $D^*(w) \cap a_1^*a_2^* \ldots a_n^*$ is regular.*

Proof. Let $U = \{a_1, a_2, \ldots, a_n\} \subseteq V$ and denote $L = D^*(w) \cap a_1^*a_2^* \ldots a_n^*$. Consider the set M of minimal vectors in $\Psi_U(L)$; according to König Lemma, this set is finite (all its elements are incomparable). It is easy to see that

$$L = \{a_1^{s_1+i_1} a_2^{s_2+i_2} \ldots a_n^{s_n+i_n} \mid i_j \geq 0,\ 1 \leq j \leq n, (s_1, s_2, \ldots, s_n) \in M\}.$$

In conclusion, L is a regular language. □

Here are some other combinatorial properties of languages $D^*(w)$.

Theorem 2.2.4 *For all strings $w \in V^*$, the following assertions hold:*

(i) $length(D^*(w)) = \{m \in N \mid m \geq length(w)\}$.

(ii) $D^*(w)$ is Parikh linear.

(iii) $(D^*(w))^+ = D^*(w)$.

(iv) $Sub(D^*(w)) = alph(w)^*$, where $alph(w)$ is the minimal alphabet V such that $w \in V^*$ and $Sub(L)$ denotes the set of all substrings of the strings in L.

Proof. Assertion (i) is obvious.

(ii) If $V = \{a_1, a_2, \ldots, a_n\}$, then $\Psi_V(D^*(w)) = \{v_0 + \sum_{i=1}^{n} v_i j_i \mid j_1, \ldots, j_n \in \mathbf{N}\}$, where $v_0 = \Psi_V(w)$ and $v_i = (0, \ldots, 0, 1, 0, \ldots, 0)$, with 1 appearing on the i-th position, for all $1 \leq i \leq n$.

(iii) Clearly

$$D^{j_1}(w)D^{j_2}(w)\ldots D^{j_m}(w) \subseteq D^{j_1+j_2+\ldots+j_m+m-1}(w)$$

holds for all $m \geq 1$ and $j_i \geq 0$, $1 \leq i \leq m$. This implies $(D^*(w))^+ \subseteq D^*(w)$. The opposite inclusion is obvious.

(iv) The assertion follows from the more general fact that for each string $z = u_1\alpha u_2\beta u_3 \in D^*(w)$, there is a string in $D^*(w)$ which contains both $\alpha\beta$ and $\beta\alpha$ as substrings, for all possible $u_1, u_2, u_3, \alpha, \beta$. Indeed, by duplicating $\alpha u_2\beta$ in z one gets the string $z' = u_1\alpha u_2\beta\alpha u_2\beta u_3$, then, by duplicating $\beta\alpha$ in this latter string, one gets

$$z'' = u_1\alpha u_2\beta\alpha\beta\alpha u_2\beta u_3,$$

which contains both $\alpha\beta$ and $\beta\alpha$ as substrings. $\square$

The third assertion of the last theorem implies the closure of the family of all languages $D^*(w)$ under Kleene $+$, while the last assertion of the same theorem implies the non-closure of this family under all other AFL operations:

union: take $D^*(ab) \cup D^*(ba)$,

concatenation: take $D^*(a)D^*(b)$,

intersection with regular sets: take $D^*(ab) \cap a^+b^+ = a^+b^+$,

morphisms: take $D^*(ab)$, $h(a) = aa$ and $h(b) = bb$,

inverse morphisms: for $h(a) = h(b) = a$ we have $h^{-1}(a^+) = \{a, b\}^+$.

Finally we remark that in this section the three operations inversion, transposition and duplication have been studied isolated from each other. Language generating devices based on different variants of duplications have been also considered, see, e.g, [89, 94].

However, if we want to model the evolution it is necessary to consider schemes which contain rules for inversion as well as for transposition, duplication and deletion. A grammatical approach in this direction is presented in the next section.

2.3 The Duplication Root

The following well-known lemma, with a very simple proof, will be very useful in this section.

Lemma 2.3.1 *The equation $uv = vx$ has the solutions $v = (\alpha\beta)^k\alpha$, $k \geq 0$, $u = \alpha\beta$, $x = \beta\alpha$.*

A *square* is an immediate repeated nonempty string, that is a string x which can be written as $x = yy$, with y a nonempty string. A string is called *square-free* if it has no square as a substring. Axel Thue was the first who studied different problems related to square-free strings, see, [128, 129].

Let V be an alphabet; for a string $w \in V^+$ we write $w \triangleright z$ if $w = uxxy$ and $z = uxy$, for some $u, y \in V^*$, $x \in V^+$. We say that z is obtained from w by reducing the duplication (square) xx. The reflexive and transitive closure of the relation $\triangleright$ is denoted by $\triangleright^*$. A square-free string z is said to be a *duplication root* of w iff $w \triangleright^* z$. It is obvious that each string has a duplication root; a natural problem concerns the uniqueness of this root and the complexity of computing this root, provided that it is unique.

Lemma 2.3.2 *Let V be an alphabet and $\alpha \in V^+$. If $\alpha \triangleright \beta$ and $\alpha \triangleright \gamma$ for some strings $\beta, \gamma \in V^+$, then there exists a string $\sigma \in V^+$ such that $\beta \triangleright^* \sigma$ and $\gamma \triangleright^* \sigma$.*

Proof. Assume that α contains two duplications which can be reduced; more precisely let xx and yy be two duplications which appear in α. Assume that β and γ are obtained from α by reducing the duplication xx and yy, respectively. We distinguish two main cases:

Case 1: The strings xx and yy do not overlap each other in α. Hence, $\alpha = uxxvyyz$, $\beta = uxvyyz$, and $\gamma = uxxvyz$, for some $u, v, y \in V^*$. We take $\sigma = uxvyz$; clearly $\beta \triangleright \sigma$ and $\gamma \triangleright \sigma$.

Case 2: The strings xx and yy do overlap each other. Several subcases are considered.

Subcase 2a: The strings xx and yy overlap each other as shown in Figure 2.

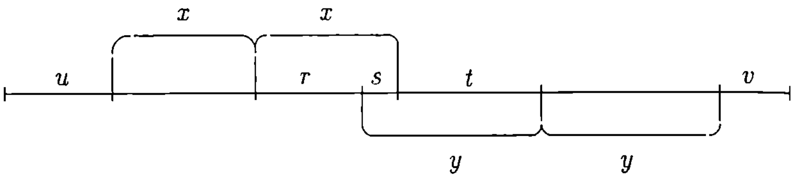

Figure 2.1.

It follows that $\beta = uxtyv$, $\gamma = uxryv$; furthermore $x = rs$ and $y = st$. Clearly,

$$\begin{aligned} \beta &= urstyv = uryyv \triangleright uryv \\ \gamma &= uxryv = uxrstv = uxxtv \triangleright uxtv = urstv = uryv \end{aligned}$$

hence the assertion holds for $\sigma = uryv$

Subcase 2b: The strings xx and yy overlap each other as shown in Figure 3.

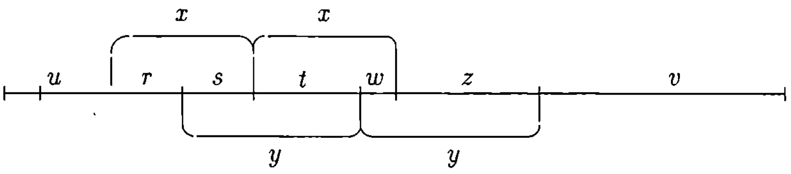

Figure 2.2.

It follows that $\beta = urszv$, $\gamma = urstv$; furthermore $rs = tw$ and $st = wz$. From the last two equalities one obtains $rsz = twz = tst$. If $t = z$, then $\beta = \gamma$ and the assertion is trivially true for $\sigma = \beta = \gamma$.

If $t \neq z$, then we firstly assume that $|z| > |t|$. Thus, $z = z't$ and $t = rt'$ for some $z', t' \in V^+$. The equation $rsz = tst$ becomes $sz' = t's$ which implies that (see Lemma 2.3.1)

$$\begin{aligned} s &= (\delta\rho)^k\delta \text{ for some } k \geq 0, \\ z' &= \rho\delta, \\ t' &= \delta\rho \end{aligned}$$

for some $\delta, \rho \in V^+$. Let us suppose that $k = 0$; equation $st = wz$ becomes $\delta r\delta\rho = w\rho\delta r\delta\rho$, which is a contradiction.

Consequently, $k \geq 1$, $t = r\delta\rho$, and $z = \rho\delta r\delta\rho$, hence

$$\beta = ur(\delta\rho)^k\delta\rho\delta r\delta\rho v \qquad \gamma = ur(\delta\rho)^k\delta r\delta\rho v$$

for some $k \geq 1$. Obviously, the assertion holds for $\sigma = \gamma$.

We now assume that $|z| < |t|$; from $rsz = tst$ one infers that $t = t'z$ and $r = tr'$. It follows that $r's = st'$ which leads to the solutions

$$\begin{aligned} s &= (\delta\rho)^k\delta \text{ for some } k \geq 0, \\ r' &= \rho\delta, \\ t' &= \delta\rho \end{aligned}$$

for some $\delta, \rho \in V^+$. Hence $t = \rho\delta z$ and $r = \rho\delta t\delta\rho$ which imply that

$$\beta = u\rho\delta z\delta\rho(\delta\rho)^k\delta zv \qquad \gamma = u\rho\delta z\delta\rho(\delta\rho)^k\delta\rho\delta zv.$$

By taking $\sigma = \beta$, the proof of this case is complete.

Subcase 2c: The strings xx and yy overlap each other as shown in Figure 4.

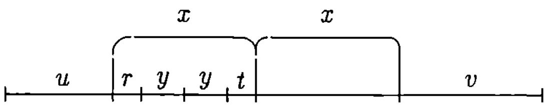

Figure 2.3.

It follows that $\beta = uxv$ and $\gamma = urytxv$ with $x = ryyt$. We have

$$\begin{aligned} \beta &= uryytv \triangleright urytv \\ \gamma &= urytryytv \triangleright urytrytv \triangleright urytv. \end{aligned}$$

We now take $\sigma = urytv$ which concludes the proof of this subcase.

Subcase 2d: The strings xx and yy overlap each other as shown in Figure 5.

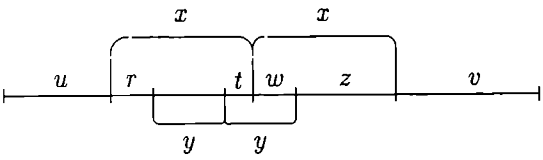

Figure 2.4.

It follows that $\beta = urytv$, $\gamma = uryzv$. Moreover, $ryt = wz$ and $y = tw$ which imply $rtwt = wz$. Assume that $z = z't$ for some $z' \in V^+$; one gets the equation $rtw = wz'$ having the solutions (see Lemma 2.3.1)

$$\begin{aligned} w &= (\delta\rho)^k\delta, k \geq 0, \\ z' &= \rho\delta, \\ rt &= \delta\rho \end{aligned}$$

for some $\delta, \rho \in V^+$. Consequently, $z = \rho\delta t$. We infer that

$$\begin{aligned} \beta &= urtwtv = u\delta\rho(\delta\rho)^k\delta tv, \\ \gamma &= urtwzv = u\delta\rho(\delta\rho)^k\delta\rho\delta tv \end{aligned}$$

for some $k \geq 0$. We take $\sigma = \beta$ which concludes the proof of this subcase.

Since any other situation can be reduced to one of those considered above, the proof is complete. □

Lemma 2.3.3 *If β is a square-free string, $\alpha \triangleright^* \beta$, and $\alpha \triangleright \gamma$, then $\gamma \triangleright^* \beta$.*

Proof. We prove this lemma by induction on n, the length of α. The case $n = 1$ is vacuously true.

We now assume that the assertion is true for any string α of length at most n and take the reduction $\alpha \triangleright \alpha' \triangleright^n \beta$ and $\alpha \triangleright \gamma$. By Lemma 2.3.2, there exists a string σ such that $\alpha' \triangleright^* \sigma$ and $\gamma \triangleright^* \sigma$. Assume that

$$\alpha' = \alpha'_1 \triangleright \alpha'_2 \triangleright \ldots \triangleright \alpha'_k = \sigma,$$

for some $k \geq 1$. By the induction hypothesis ($|\alpha'| < |\alpha|$), one can infer that $\alpha'_j \triangleright^* \beta$ for all $1 \leq j \leq k$, hence $\gamma \triangleright^* \beta$, too. □

A direct consequence of this lemma is the next result which states the uniqueness of the duplication root.

Theorem 2.3.1 *Each string has a unique duplication root.*

Moreover, in the process of finding the duplication root at any step it does not matter which duplication is reduced.

We now discuss an algorithm for finding the duplication root of a given string. To this aim, we need an algorithm for finding a duplication in a string. There were reported several algorithms for finding all duplications based on the suffix tree method [4, 22, 84]. Other algorithms can find just one duplication [85, 108]. The latter one is based on the fingerprinting method and determines the shortest duplication. All of them require $O(n \log n)$ time, where n is the length of the string. However, if the alphabet is fixed, the algorithm proposed in [85] can find a duplication in time $O(n)$.

By Lemma 2.3.3, based on these algorithms one immediately infers

Theorem 2.3.2 *1. There is an algorithm for finding the duplication root of a string y in $O(|y|^2 \log |y|)$, provided that the alphabet of y is not fixed.*

2. There is an algorithm for finding the duplication root of a string y in $O(|y|^2)$, provided that the alphabet of y is fixed.

We finish this section by an open problem: It is known that finding all duplications in a string x cannot be done in less time than $O(|x| \log |x|)$ when the alphabet is not fixed. The algorithm announced in the previous theorem is based on such an algorithm. Is this algorithm optimal as well?

2.4 Multiple Crossing-over

One was observed that the linkage between genes were never complete because of the exchange events between homologous chromosomes.

This recombination process by exchanging of segments between homologous chromosomes is called *crossing-over.*

Each gene occupies a well-defined site or locus in its chromosome, having corresponding locations in the pair of homologous chromosomes. Crossing-over has the following features:

1. The exchange of segments occurs after the chromosomes have replicated.

2. The exchange process involves a breaking and rejoining of the two chromatids, resulting in the reciprocal exchange of equal and corresponding segments between them.

3. Crossing-over occurs more or less at random along the length of a chromosome pair.

In [65], an operation on strings and languages having the same features is introduced. The operation is applicable to a pair of strings of equal length as the crossing-over between homologous chromosomes.

Each string is cut in several fragments, but in the same sites for both of them, and crossing these fragments by ligases. A new string, of the same length, is formed by starting at the left end of one parent, copying a segment, crossing over to the next site in the other parent, copying a substring, crossing back to the first parent and so on until the right end of one parent is reached. Obviously, a new string can be obtained by starting with the other parent.

Let us remark the similarity between the crossing-over on words and the chromosome crossing-over:

- the words are of the same length;

- the corresponding segments which are interchanged between them are of the same length;

- the number of sites in strings is arbitrarily large;

- the sites in strings are at random, along the strings.

In this section, we consider four crossing-over operations, which are slight generalizations of those introduced in [65] and [93], and we study the relation between these operations and other operations in formal language theory, especially the shuffle operations.

Let M be a finite subset of $V^*\#V^*\$V^*\#V^*$, $\#, \$ \notin V$, whose elements are called *crossover rules* and w_1, w_2 be two strings of equal length in V^*.

We define the relation

$$(w_1, w_2) \Longrightarrow_M^{Eql} w$$

iff the following conditions hold:

(i) $w_1 = \underbrace{t_1 x_1}_{u_1} \underbrace{x'_1 z_1 x_2}_{u_2} \underbrace{x'_2 t_3 x_3}_{u_3} \underbrace{x'_3 z_3 x_4}_{u_4} \ldots \underbrace{x'_k z_{k+1}}_{u_{k+1}}$, for some $k \geq 1$

(ii) $w_2 = \underbrace{t'_1 y_1}_{v_1} \underbrace{y'_1 z'_1 y_2}_{v_2} \underbrace{y'_2 t'_3 y_3}_{v_3} \underbrace{y'_3 z'_3 y_4}_{v_4} \ldots \underbrace{y'_k z'_{k+1}}_{v_{k+1}}$

(iii) $|u_i| = |v_i|$, $1 \leq i \leq k+1$,

(iv) $x_i \# x'_i \$ y_i \# y'_i \in M$, $1 \leq i \leq k$.

and

$$w = \begin{cases} u_1 v_2 u_3 \ldots u_i v_{i+1} \ldots u_{k+1} & \text{if } k \text{ is even,} \\ u_1 v_2 u_3 \ldots u_i v_{i+1} \ldots v_{k+1} & \text{if } k \text{ is odd} \end{cases}$$

Moreover, for two arbitrary strings w_1, w_2,

$$(w_1, w_2) \Longrightarrow_M w$$

iff the above conditions, excepting (iii), hold.

For a pair of strings (w_1, w_2) we denote

$$EqlCO_M(w_1, w_2) = \{w \mid (w_1, w_2) \Longrightarrow_M^{Eql} w\}.$$

and

$$CO_M(w_1, w_2) = \{w \mid (w_1, w_2) \Longrightarrow_M w\}.$$

For two languages L_1, L_2 over V^* we define:

(i) the (non-iterated) *equal length crossing-over*

$$EqlCO_M(L_1, L_2) = \bigcup_{w_1 \in L_1,\ w_2 \in L_2} EqlCO_M(w_1, w_2).$$

(ii) the (non-iterated) *crossing-over*

$$CO_M(L_1, L_2) = \bigcup_{w_1 \in L_1,\ w_2 \in L_2} CO_M(w_1, w_2).$$

As we can see, in the above definitions, the strings can be cut in arbitrarily many fragments. In the case of splicing operation the number of segments is limited to one and in [93] to a given integer.

We shall consider a generalization of the operation studied in [93].

Let $k \geq 1$ and M be a finite subset of $(V^*\#V^*\$V^*\#V^*)^k$, $\#, \$ \notin V$, and w_1, w_2 be two strings of equal length in V^*. Both strings are split in $k+1$ segments at the sites indicated by the rules of M. Formally, if

$$(x_1\#x'_1\$y_1\#y'_1, x_2\#x'_2\$y_2\#y'_2, \ldots, x_k\#x'_k\$y_k\#y'_k) \in M$$

and

$$\begin{aligned} w_1 &= \underbrace{t_1x_1}_{u_1}\underbrace{x'_1z_1x_2}_{u_2}\underbrace{x'_2t_3x_3}_{u_3}\underbrace{x'_3z_3x_4}_{u_4}\ldots\underbrace{x'_kz_{k+1}}_{u_{k+1}}, \\ w_2 &= \underbrace{t'_1y_1}_{v_1}\underbrace{y'_1z'_1y_2}_{v_2}\underbrace{y'_2t'_3y_3}_{v_3}\underbrace{y'_3z'_3y_4}_{v_4}\ldots\underbrace{y'_kz'_{k+1}}_{v_{k+1}} \\ |u_i| &= |v_i|,\ 1 \leq i \leq k+1 \end{aligned}$$

then

$$(w_1, w_2) \Longrightarrow_M^{k-Eql} \begin{cases} u_1v_2u_3\ldots u_iv_{i+1}\ldots u_{k+1} \text{ if } k \text{ is even,} \\ u_1v_2u_3\ldots u_iv_{i+1}\ldots v_{k+1} \text{ if } k \text{ is odd} \end{cases}$$

The relation $\Longrightarrow_M^k$ is defined by omitting the condition on the segments u_i, v_i.

As for the previous operations, for a pair of strings (w_1, w_2) we denote

$$k - EqlCO_M(w_1, w_2) = \{w \mid (w_1, w_2) \Longrightarrow_M^{k-Eql} w\}.$$

and

$$k - CO_M(w_1, w_2) = \{w \mid (w_1, w_2) \Longrightarrow_M^{k} w\}.$$

Note here that for $1 - CO$ is actually the splicing operation.

For two languages L_1, L_2 over V^* we define:

(i) the (non-iterated) *equal length k-crossing-over*

$$k - EqlCO_M(L_1, L_2) = \bigcup_{w_1 \in L_1,\ w_2 \in L_2} k - EqlCO_M(w_1, w_2).$$

(ii) the (non-iterated) *k-crossing-over*

$$k - CO_M(L_1, L_2) = \bigcup_{w_1 \in L_1,\ w_2 \in L_2} k - CO_M(w_1, w_2).$$

Remember that a trio is a family of languages closed under ε-free homomorphisms, inverse homomorphisms, and intersection with regular sets. A full trio is a trio closed under arbitrary homomorphisms.

Lemma 2.4.1 *If a trio is closed under $SShuf$ operation, then it is closed under $EqlCO$.*

Proof. Consider an alphabet V, two languages L_1, L_2 over V, in a trio denoted by F, and a (finite) set of rules $M = \{x_i\#y_i\$z_i\#t_i | 1 \leq i \leq m\}$.

For any character $a \in V$, we consider a new symbol $a' \notin V$ and denote $V' = \{a' \mid a \in V\}$. Consider also $2m$ new symbols $c_i, c'_i, 1 \leq i \leq m$.

In the following, by a primed word (language) we shall understand that all symbols of the original word (language) are replaced by primed symbols. For instance, if $w = a_1 a_2 \dots a_n \in V^*$, then $w' = a_1' a_2' \dots a_n'$ and if $A \subseteq V^*$, then $A' = \{w' \mid w \in A\}$.

Take the homomorphisms

$$\begin{array}{lll}
h_1 & & (V \cup \{c_i \mid 1 \le i \le m\})^* \longrightarrow V^*, \\
 & & h_1(a) = a, \text{ for any } a \in V, \\
 & & h_1(c_i) = x_i y_i, \text{ for any } 1 \le i \le m, \\
h_2 & : & (V \cup \{c_i \mid 1 \le i \le m\})^* \longrightarrow (V \cup \{c_i \mid 1 \le i \le m\})^*, \\
 & & h_2(a) = a, \text{ for any } a \in V, \\
 & & h_2(c_i) = x_i c_i y_i, \text{ for any } 1 \le i \le m, \\
h_3 & & (V' \cup \{c_i' \mid 1 \le i \le m\})^* \longrightarrow V^*, \\
 & & h_3(a') = a, \text{ for any } a' \in V', \\
 & & h_3(c_i') = z_i t_i, \text{ for any } 1 \le i \le m, \\
h_4 & & (V' \cup \{c_i' \mid 1 \le i \le m\})^* \longrightarrow (V' \cup \{c_i' \mid 1 \le i \le m\})^*, \\
 & & h_4(a') = a', \text{ for any } a' \in V', \\
 & & h_4(c_i') = z_i' c_i' t_i', \text{ for any } 1 \le i \le m.
\end{array}$$

The language

$$\bar{L} = SShuf(h_2(h_1^{-1}(L_1)), h_4(h_3^{-1}(L_2))) \cap (\bigcup_{i=1}^{m} (VV')^* \{c_i c_i'\})^* (VV')^*$$

is in F. A string in $\bar{L}$ has the form

$$a_{p_{1,1}} a'_{q_{1,1}} a_{p_{1,2}} a'_{q_{1,2}} \dots a_{p_{1,n_1}} a'_{q_{1,n_1}} c_{i_1} c'_{i_1} a_{p_{2,1}} a'_{q_{2,1}} \dots a_{p_{2,n_2}} a'_{q_{2,n_2}} c_{i_2} c'_{i_2} \dots \\ c_{i_k} c'_{i_k} a_{p_{k+1,1}} a'_{q_{k+1,1}} \dots a_{p_{k+1,n_{k+1}}} a'_{q_{k+1,n_{k+1}}},$$

where, for any $1 \le r \le k+1, 1 \le s \le n_r, a_{p_{r,s}} \in V, a'_{q_{r,s}} \in V'$, and, for any $2 \le r \le k+1$, there are some $m_r, l_r, d_r, e_r,\ 1 \le m_r, l_r, d_r, e_r \le n_r$, such that

$$\begin{aligned}
a_{p_{r,1}} a_{p_{r,2}} \dots a_{p_{r,m_r}} &= y_{i_{r-1}}, \\
a'_{q_{r,1}} a'_{q_{r,2}} \dots a'_{q_{r,l_r}} &= t'_{i_{r-1}}, \\
a_{p_{r-1,d_r}} a_{p_{r-1,d_{r+1}}} \dots a_{p_{r-1,n_{r-1}}} &= x_{i_{r-1}}, \\
a'_{q_{r-1,e_r}} a'_{q_{r-1,e_{r+1}}} \dots a'_{q_{r-1,n_{r-1}}} &= z'_{i_{r-1}}.
\end{aligned}$$

Construct an ε-free gsm g which leads a string in L_1 as above to the string

$$a_{p_{1,1}} a'_{q_{1,1}} a_{p_{1,2}} a'_{q_{1,2}} \dots a_{p_{1,n_1}} a'_{q_{1,n_1}} c_{i_1} c'_{i_1} a'_{p_{2,1}} a_{q_{2,1}} a'_{p_{2,2}} a_{q_{2,2}} \dots$$
$$a'_{p_{2,n_2}} a_{q_{2,n_2}} c_{i_2} c'_{i_2} a_{p_{3,1}} a'_{q_{3,1}} \dots a_{p_{3,n_3}} a'_{q_{3,n_3}} c_{i_3} c'_{i_3} a'_{p_{4,1}} a_{q_{4,1}} \dots .$$

In order to do that task, g works as follows:

- start scanning the string in the state s_0,
- leave unchanged all symbols $a_{p_{r,s}}, a'_{q_{r,s}}$ until either the right end of the string is detected and, in this case, **stop**, or the next c_i is reached,
- leave c_i and c'_i unchanged but change the state into s_1,
- change any $a_{p_{r,s}}$ into $a'_{p_{r,s}}$ and any $a'_{q_{r,s}}$ into $a_{q_{r,s}}$ until either the right end of the string is detected and, in this case, **stop**, or the next c_i is reached,
- leave c_i and c'_i unchanged but change the state into s_0,
- restart the work from the second step for the remained part of the initial string.

Remark that both states of g are final.
Consider now the homomorphism

$$h_5 : (V \cup V' \cup \{c_i \mid 1 \le i \le m\} \cup \{c'_i \mid 1 \le i \le m\})^* \longrightarrow V^*,$$
$$h_5(a) = a, \text{ for any } a \in V,$$
$$h_5(a') = \varepsilon, \text{ for any } a' \in V',$$
$$h_5(c_i) = h_5(c'_i) = \varepsilon, \text{ for any } 1 \le i \le m.$$

Obviously, we have

$$EqlCO_M(L_1, L_2) = h_5(g(\bar{L})).$$

Because any trio is closed under ε-free gsm mappings, it follows that the language $g(\bar{L})$ is in F. As it could be easily seen, h_5 is 4-limited erasing on $g(\bar{L})$. Consequently, $EqlCO_M(L_1, L_2)$ is in the considered trio and the proof is over. □

Lemma 2.4.2 *Any trio closed under $EqlCO$ operation is closed under $SShuf$ operation.*

Proof. Let $L_1 \subseteq V_1^*, L_2 \subseteq V_2^*$ be two languages in a trio F. Consider the alphabets

$$\begin{aligned} \bar{V}_1 &= \{a_1, a_2, a_3 | a \in V_1\}, \\ V_2' &= \{a' \mid a \in V_2\}, \\ \bar{V}_2' &= \{a_1, a_2, a_3 | a \in V_2'\} \end{aligned}$$

and the following homomorphisms:

$$\begin{aligned} &h_1 : V_1^* \longrightarrow \bar{V}_1^*, \quad h_1(a) = a_1a_2a_3, \text{ for any } a \in V_1, \\ &h_2 : V_2'^* \longrightarrow \bar{V}_2'^*, \quad h_2(a') = a_1'a_2'a_3', \text{ for any } a' \in V_2', \\ &h_3 : V_2^* \longrightarrow V_2'^*, \quad h_3(a) = a', \text{ for any } a \in V_2, \end{aligned}$$

$$\begin{aligned} h_4 \quad : \quad & (\bar{V}_1 \cup \bar{V}_2')^* \longrightarrow (V_1 \cup V_2)^*, \\ & h_4(a_1) = a, \ h_4(a_2) = h_4(a_3) = \varepsilon, \text{ for any } a \in V_1, \\ & h_4(a_2') = a, \ h_4(a_1') = h_4(a_3') = \varepsilon, \text{ for any } a' \in V_2'. \end{aligned}$$

Consider also the set of rules $M = \{a_1\#a_2\$b_1'\#b_2' | a \in V_1, b \in V_2\} \cup \{a_3\#\varepsilon\$b_3'\#\varepsilon | a \in V_1, b \in V_2\}$. In these conditions, it is easy to check that

$$SShuf(L_1, L_2) = h_4(EqlCO_M(h_1(L_1), h_2(h_3(L_2)))) \cap (\bar{V}_1(\bar{V}_2')^2)^*).$$

Mentioning that the homomorphism h_4 is 1-limited erasing we conclude the proof. □

Theorem 2.4.1 *A trio is closed under $SShuf$ if and only if it is closed under $EqlCO$.*

We mention now a known result concerning the interdependence between the $SShuf$ and $Shuf$ operations in the frame of a trio, which will turn to be very useful in the sequel.

Theorem 2.4.2 *Any trio is closed under $SShuf$ if and only if it is closed under $Shuf$.*

Theorem 2.4.3 *Any full trio closed under the operation $Shuf$ is closed under the operation CO.*

Proof. By using the same construction as in Lemma 2.4.1, the language

$$\bar{L} = Shuf(h_2(h_1^{-1}(L_1)), h_4(h_3^{-1}(L_2))) \cap (\bigcup_{i=1}^{m}(V^*V'^*)\{c_i c_i'\})^*(V^*V'^*)$$

is in the considered trio, if L_1 and L_2 are in the trio.

Take the *gsm*, g working as follows:

- start scanning the string in the state s_0,
- leave unchanged all non-primed symbols until a primed symbol is reached,
- remove all primed symbols until a symbol c_i is detected,
- change the state into s_1, erase the letters c_i, c_i' and replace all primed symbols by non-primed symbols,
- remove all non-primed symbols until the next symbol c_j is detected
- change the state into s_0 and erase the letters c_j, c_j'
- go on until the end of the string is reached

Therefore, $CO_M(L_1, L_2) = g(\bar{L})$ and the proof is over. □

Theorem 2.4.4 *Any full trio closed under $EqlCO$ is closed under CO.*

Proof. The proof is a consequence of all the previous theorems. □

Next, we shall examine the case of the prescribed number of crossing-overs.

Theorem 2.4.5 *Any full trio closed under $Shuf$ is closed under $k-CO$, for any $k \geq 1$.*

Proof. The proof is a slight modification of the proof of Lemma 2.4.2. Let k be a positive integer and L_1, L_2 be two languages over V in a trio F; assume that

$$M = \{(x_{i_1}\#y_{i_1}\$z_{i_1}\#t_{i_1}, x_{i_2}\#y_{i_2}\$z_{i_2}\#t_{i_2}, \ldots, x_{i_k}\#y_{i_k}\$z_{i_k}\#t_{i_k})| \\ 1 \leq i \leq n\}.$$

Consider kn new symbols $c_{i_j}, 1 \leq i \leq n, 1 \leq j \leq k$, and the homomorphisms defined as follows:

$h_1 : (V \cup \{c_{i_j} \mid 1 \leq i \leq n, 1 \leq j \leq k\})^* \to V^*$,
$h_1(a) = a, a \in V$,
$h_1(c_{i_j}) = x_{i_j}y_{i_j}, 1 \leq i \leq n, 1 \leq j \leq k.$

$h_2 : (V \cup \{c_{i_j} \mid 1 \leq i \leq n, 1 \leq j \leq k\})^* \to (V \cup \{c_{i_j} \mid 1 \leq i \leq n, 1 \leq j \leq k)^*$
$h_2(a) = a$,
$h_2(c_{i_j}) = x_{i_j}c_{i_j}y_{i_j}, 1 \leq i \leq n, 1 \leq j \leq k\}.$

The language

$$L_3 = h_2(h_1^{-1}(L_1)) \cap E$$

where E is the regular language

$$E = \bigcup_{i=1}^{n} V^*\{c_{i_1}\}V^*\{c_{i_2}\}V^* \ldots V^*\{c_{i_k}\}V^*$$

is in F.

Analogously, the language

$$L_4 = h_4(h_3^{-1}(L_2')) \cap \bigcup_{i=1}^{n} V'^*\{c'_{i_1}\}V'^*\{c'_{i_2}\}V'^* \ldots V'^*\{c'_{i_k}\}V'^*$$

is also in F, where the homomorphisms h_3 and h_4 work as h_1 and h_2, respectively, but on primed symbols and strings.

From the closure properties of the family F we have that

$$L_5 = Shuf(L_3, L_4) \cap \bigcup_{i=1}^{n} V^*V'^*\{c_{i_1}c'_{i_1}\}V^*V'^* \dots V^*V'^*\{c_{i_k}c'_{i_k}\}V^*V'^*$$

is in F.

We have $k - CO_M(L_1, L_2) = g(L_5)$, where g is a *gsm* similar to that constructed in Theorem 2.4.3. Because any full trio is closed under arbitrary gsm mappings it follows that $k - CO_M(L_1, L_2) \in F$. □

Theorem 2.4.6 *If F is a full trio closed under union and $EqlCO$ operation, then F is closed under $k - CO$.*

Proof. It follows from the previous theorem and Theorem 2.4.2. □
By combining the ideas used for proving Lemma 2.4.1 and Theorem 2.4.5 one can get

Theorem 2.4.7 *Any trio closed under $SShuf$ is closed under $k - EqlCO$, for any $k \geq 1$.*

and, consequently

Corollary 2.4.1 *Any trio closed under $EqlCO$ is closed under $k - EqlCO$, for any $k \geq 1$.*

We finish this section by pointing out some further directions of research and open problems. Several natural questions which can naturally arise, if we are looking to the above diagram, are:

1. Are there trios closed under $k - EqlCO$ and not closed under $SShuf$, for different values of k ?
2. Are there trios closed under CO and not closed under $Shuf$?
3. Are there trios closed under $k - CO$ and not closed under $Shuf$, for different values of k ?

As far as the last question is concerned, a partial answer is

Theorem 2.4.8 *There are trios closed under* $1 - CO$ *and* $2 - CO$ *but not closed under* $Shuf$.

Proof. Take the class of context-free languages which is a trio and is closed under splicing $(1 - CO)$ [105] but it is not closed under $Shuf$.

On the other hand, the family of context-free languages is closed under $2 - CO$. Let L_1, L_2 be two context-free language over V and M be a set of 2-crossover rules. Assume that

$$M = \{(x_{i_1}\#x_{i_2}\$y_{i_1}\#y_{i_2}, x_{i_3}\#x_{i_4}\$y_{i_3}\#y_{i_4})| \\ 1 \leq i \leq n\}.$$

Consider $4n$ new symbols $c_{i,j}$, and $c'_{i,j}, 1 \leq i \leq n, j = 1, 2$ and the homomorphisms

$$h_1 : (V \cup \{c_{i,j} \mid 1 \leq i \leq n, j = 1,2\})^* \to V^*,$$

$$h_1(a) = a, a \in V,$$
$$h_1(c_{i,j}) = x_{i_{2j-1}} x_{i_{2j}},$$

$$h_2 : (V \cup \{c'_{i,j} \mid 1 \leq i \leq n, j = 1,2\})^* \to V^*,$$

$$h_2(a) = a, a \in V,$$
$$h_2(c'_{i,j}) = y_{i_{2j-1}} y_{i_{2j}}.$$

The languages

$$L_3 = h_1^{-1}(L_1) \cap \bigcup_{i=1}^{n} V^*\{c_{i,1}\}V^*\{c_{i,2}\}V^*$$

$$L_4 = h_2^{-1}(L_2) \cap \bigcup_{i=1}^{n} V^*\{c'_{i,1}\}V^*\{c'_{i,2}\}V^*$$

are context-free.

For a language L denote by $Sub(L)$ the set of all subwords of the words in L. Define the substitution

$$s : (V \cup \{c_{i,j} \mid 1 \leq i \leq n, j = 1, 2\})^* \to 2^{(V \cup \{c_{i,j},\ c'_{i,j} | 1 \leq i \leq n,\ j=1,2\})^*},$$

$$s(a) = a, a \in V,$$
$$s(c_{i,2}) = \{c_{i,2}\}, 1 \leq i \leq n,$$
$$s(c_{i,1}) = c'_{i,1}V^*c'_{i,2} \cap Sub(L_4), 1 \leq i \leq n.$$

The above substitution is a substitution with context-free languages. Indeed, $Sub(L_4)$ is a context-free language and $c'_{i,1}V^*c'_{i,2}$ are regular languages for all $1 \leq i \leq n$. Under these circumstances, we conclude that $s(L_3)$ is context-free. Now, the words in $s(L_3)$ are of the form

$$w = uc'_{i,1}vc'_{i,2}tc_{i,2}z$$

for some $1 \leq i \leq n$, and $u, v, t, z \in V^*$.

We construct a gsm g which replaces the symbol $c'_{i,1}$ by x_{i_1}, removes all symbols after $c'_{i,1}$ until $c_{i,2}$, and replaces $c_{i,2}$ by x_{i_4}, for all $1 \leq i \leq n$. It follows that $2 - CO_M(L_1, L_2) = g(s(L_3))$ therefore, the class of context-free languages is closed under $2 - CO$. □

For all operations considered here one can define, in a natural way, the iterated case. What are the interdependence relations between them in this case ?

2.5 Two Crossover Distances

A basic problem in the area of combinatorial algorithms for genome evolution is to determine the minimum number of large scale evolutionary events (genome rearrangements) that transform a genome into another. The present section, based on [95], is a contribution to the algorithmic study of genom evolution by crossovers (translocations).
Two types of crossover distance between two sets of strings (genomes) are introduced; we examine the complexity of computing these distances in the case of uniform crossover, that is at each step the strings exchange prefixes of the same length. We present exact polynomial algorithms based on the "greedy" strategy when the target set is a singleton. When considering arbitrary target sets a 2-approximation algorithm is provided for computing the sequential crossover distance. Some open problems are also formulated.

Prior work dealing with the combinatorial analysis of genome operations has focused on evolution distance in terms of inversions, transpositions or crossovers for chromosomes formed from different markers which correspond to unique segments of DNA. From the formal point of view this means that all symbols of the strings representing the chromosomes are different. Thus Kececioglu and Sankoff [66], [67] developed exact and approximation algorithms for two types of inversion distance, Bafna and Pevzner reported approximation algorithms for transposition distance [8]. More recently [68], Kececioglu and Ravi discussed exact and approximation algorithms for distance involving crossovers alone as well as together with inversions. Some applications of these results to biological data are now underway [9], [55].

Our work differs from these approaches in many respects: the strings representing chromosomes may have multiple occurrences of the same symbol, they may have common symbols, the number of copies of all strings in the initial set is considered to be arbitrarily large, the definition of the crossover distance.

Let V be a given alphabet (practically this alphabet is the DNA alphabet $\{A, T, C, G\}$); chromosomes may be viewed as strings over this alphabet. For each string $x \in V^+$, $x[i,j]$ delivers the substring of x that starts at position i and ends at position j in x, $1 \leq i \leq j \leq |x|$. Conventionally, $x[i,j]$ is the empty string in all cases $j < i$.

For two strings x, y over an alphabet V and two integers $1 \leq i < |x|, 1 \leq j < |y|$, we define the crossover operation

$$\begin{aligned}&(x,y) \vdash_{(i,j)} (z_1, z_2) \text{ iff } x = tu, y = vw, z_1 = tw, z_2 = vu,\\ &\text{and } |t| = i, |v| = j.\end{aligned}$$

The pair of natural numbers (i,j) indicates the length of the prefixes they interchange with each other. When we are not interested about the length of these segments, we write simply $\vdash$. Let us note that, from a chromosome and its replica, say xyz, one may get two other chromosomes $xyyz$ and xz. It is worth mentioning here that this type of recombination is known as crossover between "sister" chromatids and it is the main way of producing tandem repeats or block deletions in chromosomes.

We extend the crossover operation to a finite set of strings $A \subseteq V^+$ by

$$CO(A) = \bigcup_{x,y \in A} \{z, w | (x,y) \vdash (z,w)\}.$$

Let A be a finite set of strings that appear arbitrarily many times in A. Define, iteratively

$$\begin{aligned} CO_0(A) &= A, \\ CO_{k+1}(A) &= CO_k(A) \cup CO(CO_k(A)), \\ CO_*(A) &= \bigcup_{k \geq 0} CO_k(A). \end{aligned}$$

A *crossover sequence* in $CO_*(A)$ is a sequence $S = s_1, s_2, \ldots, s_n$, where for each $1 \leq i \leq n$ $s_i = (x_i, y_i) \vdash_{(k_i,p_i)} (u_i, v_i)$, for some $x_i, y_i, u_i, v_i \in CO_*(A)$ and $1 \leq k_i < |x_i|$, $1 \leq p_i < |y_i|$. Given a crossover sequence as above S and as above and $x \in CO_*(A)$ we define

$$\begin{aligned} P_i(S,x) &= card\{j \leq i | x = x_j \text{ or } x = y_j\} + card\{j \leq i | x_j = y_j = x\}, \\ F_i(S,x) &= \begin{cases} \infty, \text{ if } x \in A, \\ card\{j \leq i | u = x_j \text{ or } v = y_j\} + \\ card\{j \leq i | u_j = v_j = x\} \end{cases} \end{aligned}$$

The *length* of a crossover sequence $S = s_1, s_2, \ldots, s_n$ is denoted by $lg(S)$ and equals n. A crossover sequence S as above is *contiguous* iff the following two conditions are satisfied:

(i) $x_1, y_1 \in A$,

(ii) $F_{i-1}(S, x_i) > P_{i-1}(S, x_i)$, and $F_{i-1}(S, y_i) > P_{i-1}(S, y_i)$, for all $1 \leq i \leq n$.

The second condition is very natural if one considers that the copies of the two strings that exchange prefixes are not available anymore for further crossover steps; it claims that at each crossover step at least one copy for any of the two strings involved in this step is available. By CCS we mean a contiguous crossover sequence. Let B

be a finite subset of $CO_*(A)$; a CCS S as above is B-producing if $F_n(S,z) > P_n(S,z)$ for all $z \in B$. In other words, S is B-producing if at the end of all crossover steps form S we have at least one copy at each string in B. Roughly speaking, the *sequential crossover distance* from A to B ($SCOD(A,B)$ shortly) is defined as the minimal number of steps strictly necessary to get B starting from A, providing that at each step just one crossover takes place. Formally,

$$SCOD(A,B) = \min\{lg(S) | S \text{ is a } B- \text{ producing } CCS \text{ in } CO_*(A)\}.$$

The *parallel crossover distance* from A to B ($PCOD(A,B)$, shortly) is defined as the minimal number of steps strictly necessary to get B starting from A, provided that at each step all possible crossovers take place. Formally,

$$PCOD(A,B) = \max\{\min\{k | x \in CO_k(A)\} | x \in B\}.$$

Example 2.5.1. *Let us consider the initial set* $A = \{x_1, x_2, x_3, x_4\}$ *with*

$$x_1 = abcbad, \quad x_2 = bbabd, \quad x_3 = accbabd, \quad x_4 = aaab,$$

and

$z_1 = bbcbad$	$z_2 = ababd$	$z_3 = ababad$	$z_4 = bbcbd$
$z_5 = abbababd$	$z_6 = aabad$	$z_7 = abababd$	$z_8 = bbd$
$z_9 = bbbd$	$z_{10} = bbabad$	$z_{11} = bbbabad$	$z_{12} = bbababd$
$z_{13} = bababd$	$z_{14} = accbd$	$z_{15} = bbccbabd$	$z_{16} = aababd$
$z_{17} = abcccbabd$	$z_{18} = abad$		

We provide below a B-producing CCS, $B = \{z_4, z_6, z_8, z_{11}, z_{15}, z_{16}, z_{18}\}$.

$$\begin{aligned}
&(x_1, x_2) \vdash_{(2,2)} (z_2, z_1), \ (z_1, z_2) \vdash_{(4,4)} (z_4, z_3),\\
&(x_1, x_2) \vdash_{(2,2)} (z_2, z_1), \ (z_2, x_2) \vdash_{(4,2)} (z_7, z_8),\\
&(z_3, z_7) \vdash_{(2,1)} (z_5, z_6), \ (x_2, x_3) \vdash_{(3,3)} (z_{12}, z_{14}),\\
&(z_8, z_{12}) \vdash_{(2,5)} (z_9, z_{10}), \ (x_2, x_3) \vdash_{(3,3)} (z_{12}, z_{14}),\\
&(x_2, x_3) \vdash_{(3,3)} (z_{12}, z_{14}), (z_{12}, z_{10}) \vdash_{(2,1)} (z_{11}, z_{13}),\\
&(z_{12}, x_3) \vdash_{(2,1)} (z_{15}, z_{16}), \ (x_1, x_3) \vdash_{(3,1)} (z_{17}, z_{18}).
\end{aligned}$$

Sometimes we refer to B as a target set. In the sequel we are dealing with the complexity of computing the crossover distances defined above for the case of uniform crossover i.e. all strings exchange prefixes of equal length. We distinguish two cases depending on the cardinality of target sets: singleton target sets and arbitrary target sets.

2.5.1 Singleton Target Sets

As we said above, by uniform crossover we mean a special type of crossover so that prefixes which are to be exchanged are of the same length. Formally, the crossover operation $\vdash_{(i,j)}$ is said to be uniform iff $i = j$, so that we shall simply write $\vdash_i$..

In the case of uniform crossover with a singleton target set, we may assume that the initial set of strings contains only strings of the same length, that is the length of the target string.

Lemma 2.5.1. *Let A be a given finite set of strings and z be a string of length k. Consider*

$$\begin{aligned}\bar{A} &= \{x \in A \mid |x| = k\} \cup \{x[1, |x|-1]\$^{k-|x|+1} \mid x \in A, |x| < k\} \cup \\ &\quad \{x[1, k-1]\$ \mid x \in A, |x| > k\},\end{aligned}$$

where \$ is a new symbol. Then $SCOD(A, z) = SCOD(\bar{A}, z)$ and $PCOD(A, z) = PCOD(\bar{A}, z)$.

Proof. Clearly, one can construct a z-producing CCS in $CO_*(A)$, starting from a z-producing CCS in $CO_*(\bar{A})$, of the same length, hence $SCOD(A, z) \leq SCOD(\bar{A}, z)$. Conversely, given a z-producing CCS in $CO_*(A)$, one can construct a z-producing CCS in $CO_*(\bar{A})$ of a smaller length. Consequently, $SCOD(A, z) = SCOD(\bar{A}, z)$.

The proof is complete because we note that all strings of length k that do not contain the symbol \$ from $CO_i(A)$ are in $CO_i(\bar{A})$, for all $i \geq 0$, and vice versa. □

In conclusion, throughout this section the strings in the initial set and the target string will be all of the same length.

Suppose that $A = \{x_1, x_2, \ldots, x_n\}$ and let z be an arbitrary string of length k; the following measure will be very useful in the sequel:

$$MaxSubLen(A, z, p) = \max\{q \mid \text{there exists } 1 \leq i \leq n$$

$$\text{such that } x_i[p, p+q-1] = z[p, p+q-1]\}.$$

Note that with uniform crossover, a letter at position i in a string remains at position i after moving to another string. Assume that $z \in CO_*(A)$; define iteratively the set $H(A, z)$ of intervals of natural numbers as follows:

1. $H(A, z) = \{[1, MaxSubLen(A, z, 1)]\}$;
2. Take the interval $[i, j]$ having the largest j; if $j = k$, then stop, otherwise put into $H(A, z)$ the new interval $[j+1, j + MaxSubLen(A, z, j+1)]$.

Note that we allow intervals of the form $[i, i]$ for some i to be in $H(A, z)$; moreover, for each $1 \leq i \leq k$ there are $1 \leq p \leq q \leq k$ (possibly the same) such that $i \in [p, q] \in H(A, z)$.

Lemma 2.5.2. *Let S be a z-producing CCS in $CO_*(A)$. Then, $lg(S) \geq card(H(A, z)) - 1$.*

Proof. We prove this assertion by induction on the length k of z. For $k = 1$ the assertion is trivially true because z must be in A, hence $H(A, z)$ contains just one element. Assume that the assertion is true for any string shorter than k. Let us consider a CCS $S = s_1, s_2, \ldots, s_q$ in $CO_*(A)$ producing z. Moreover, we may assume that $s_i = (x_i, y_i) \vdash_{p_i} (u_i, v_i)$, $1 \leq i \leq q$, and z has been obtained in S at the last step, that is either $u_q = z$ or $v_q = z$. Let

$$\begin{aligned} A' &= \{x[MaxSubLen(A, z, 1) + 1, k] \mid x \in A\}, \\ z' &= z[MaxSubLen(A, z, 1) + 1, k]. \end{aligned}$$

For simplicity denote $r = MaxSubLen(A, z, 1)$. Clearly, $H(A', z') = \{[i - r, j - r] \mid [i, j] \in H(A, z) \setminus \{[1, r]\}\}$, hence $card(H(A', z')) = card(H(A, z)) - 1$. Starting from S we construct a CCS in $CO_*(A')$, producing z' $S' = s'_1, s'_2, \ldots s'_m$ in the way indicated by the following procedure:

Algorithm 2.5.1.

```
Procedure Construct_CCS(S,r);
begin
$m := 0$;
for $i$:=1 to $q$ begin
  if $(p_i > r)$ then
    $m := m + 1$;
    $s'_m = (x_i[r+1,k], y_i[r+1,k]) \vdash_{p_i - r} (u_i[r+1,k], v_i[r+1,k])$;
  endif;
endfor;
end.
```

Claim 1: *S' is a CCS.*

Proof of the claim. Firstly, we note that for each $1 \leq i \leq q$ so that $p_i \leq r$, the relations $u_i[r+1,k] = y_i[r+1,k]$ and $v_i[r+1,k] = x_i[r+1,k]$ hold. Assume that $p_{i_1}, p_{i_2}, \ldots, p_{i_m}$ are all integers from $\{p_1, p_2, \ldots, p_q\}$ bigger than r. Because all $p_1, p_2, \ldots, p_{i_1} - 1$ equal at least r, it follows that both $x_{i_1}[r+1,k], y_{i_1}[r+1,k]$ are in A'.

Now, it suffices to prove that for a given $2 \leq j \leq m$, the relations

$$\begin{aligned}
F_{j-1}(S', x_{i_j}[r+1,k]) &> P_{j-1}(S', x_{i_j}[r+1,k]),\\
F_{j-1}(S', y_{i_j}[r+1,k]) &> P_{j-1}(S', y_{i_j}[r+1,k]),
\end{aligned}$$

hold. We shall prove the first relation only. It is not hard to see that

$$\begin{aligned}
F_{j-1}(S', x_{i_j}[r+1,k]) &= \sum_{x[r+1,k] = x_{i_j}[r+1,k]} F_{i_j-1}(S,x) - \\
&\quad card(X) - card(Y),\\
P_{j-1}(S', x_{i_j}[r+1,k]) &= \sum_{x[r+1,k] = x_{i_j}[r+1,k]} P_{i_j-1}(S,x) - \\
&\quad card(Z) - card(W),
\end{aligned}$$

where

$$\begin{aligned}
X &= \{t \leq i_j - 1 | p_t \leq r,\ u_t[r+1,k] = v_t[r+1,k] = x_{i_j}[r+1,k]\},\\
Y &= \{t \leq i_j - 1 | p_t \leq r,\ u_t[r+1,k] = x_{i_j}[r+1,k]\\
&\quad \text{or } v_t[r+1,k] = x_{i_j}[r+1,k]\},
\end{aligned}$$

$$\begin{aligned} Z &= \{t \le i_j - 1 | p_t \le r,\ x_t[r+1,k] = y_t[r+1,k] = x_{i_j}[r+1,k]\}, \\ W &= \{t \le i_j - 1 | p_t \le r,\ x_t[r+1,k] = x_{i_j}[r+1,k] \\ &\quad \text{or } y_t[r+1,k] = x_{i_j}[r+1,k]\}. \end{aligned}$$

But, as we have seen

$$X = Z \text{ and } Y = W.$$

In conclusion, as S is a CCS, it follows that $F_{j-1}(S', x_{i_j}[r+1,k]) > P_{j-1}(S', x_{i_j}[r+1,k])$, and the proof of the claim is complete.

Claim 2: *S' is z-producing.*

Proof of the claim. More generally, we shall prove by induction on i that S' is producing $u_i[r+1,k]$ and $v_i[r+1,k]$ for all $1 \le i \le q$. The assertion is trivially true for $i = 1$. Assume that the assertion is true for all $t \le i$; we shall prove it for $i+1$. If $u_{i+1}[r+1,k]$ is in A' or $p_{i+1} > r$, we are done. If $p_{i+1} \le r$, then $u_{i+1}[r+1,k] = y_{i+1}[r+1,k]$; for $y_{i+1} \notin A$ we have $F_i(S, y_{i+1}) > 0$, hence there exists $k \le i$ such that $u_t = y_{i+1}$ or $v_t = y_{i+1}$. By the induction hypothesis, $u_t[r+1,k]$ holds, which concludes the proof of the second claim.

But there exists at least one i such that $p_i \le r$, it follows that $m \le q-1$. By the induction hypothesis, $m \ge card(H(A', z')) - 1$, and the proof is complete. □

The next result is a direct consequence of this lemma.

Theorem 2.5.1. *Let z be a string of length k and A be a set of cardinality n. There is an exact algorithm that computes SCOD(A,z) in O(kn) time and O(kn) space.*

Proof. The following algorithm indicates how to construct a CCS $S = s_1, s_2 dots, s_m$ in $CO_*(A)$ producing z, when $z \notin A$, whose length is exactly $card(H(A,z)) - 1$.

Algorithm 2.5.2.

```
    Procedure Uniform_crossover_CCS_construction(A,z);
    begin
    $p := MaxSubLen(A, z, 1)$; let $x$ be a string in $A$ with $x[1,p] = z[1,p]$;
    $m := 0$;
    while $p < k$ begin
    r := MaxSubLen(A, z, p + 1);
    if r = 0 then THE STRING z CANNOT BE OBTAINED FROM A; stop
      else
        let y be a string in A with y[p + 1, p + r] = z[p + 1, p + r];
        m := m + 1
        s_m = (x, y) ⊢_p (u, v)};
        p:=p+r;
        x:=u;
    endif
  endwhile;
  end.
```

It is easy to see that if the algorithm successfully terminates, then either u or v is exactly z, and the length of the CCS determined by the algorithm is exactly $card(H(A,z)) - 1$. By the previous lemma, this in an optimal value. As one can easily see the time complexity of this algorithm is given by the complexity of computing the values $MaxSubLen(A,z,p)$, which is $O(kn)$. Obviously, it requires $O(kn)$ memory. □

We shall proceed to a similar approach for computing the parallel crossover distance from A to z. For a positive real number r denote by $\lceil r \rceil$ the natural number that satisfies $\lceil r \rceil - 1 < r \leq \lceil r \rceil$.

Theorem 2.5.2. *Let z be a word in $CO_*(A)$. Then*

$$PCOD(A,z) = \lceil log_2(card(H(A,z))) \rceil.$$

Proof. Denote by $q = \lceil \log_2(card(H(A,z))) \rceil$. For the beginning we prove that $z \in CO_q(A)$. The argument is an induction on q. If $q = 0$, then $card(H(A,z)) = 1$, that is $z \in A = CO_0(A)$. Assume that the assertion is true for any set A and $z \in CO_*(A)$ such that $\lceil \log_2(card(H(A,z))) \rceil < q$ and let

$$H(A,z) = \{[1,r_1],[r_1+1,r_2],\ldots,[r_{p-1}+1,k]\}$$

with $2^{q-1} < p \leq 2^q$. Consider $B = CO_1(A)$; it is easy to see that

$$H(B,z) = \begin{cases} \{[1,r_2],[r_2+1,r_4],\ldots,[r_{p-2}+1,k]\}, & \text{if } p \text{ is even,} \\ \{[1,r_2],[r_2+1,r_4],\ldots,[r_{p-1}+1,k]\}, & \text{if } p \text{ is odd.} \end{cases}$$

For $\lceil \log_2(card(H(B,z))) \rceil = q-1$ it follows that $z \in CO_{q-1}(B) = CO_q(A)$, and the proof is complete. In conclusion, $PCOD(A,z) \leq \lceil \log_2(card(H(A,z))) \rceil$.

By a similar reasoning one can prove that if $z \in CO_r(A)$, then r is at least $\lceil \log_2(card(H(A,z))) \rceil$. □

Based on the previous lemma and Theorem 2.5.1 one may state:

Theorem 2.5.3. *Let z be a string of length k and A be a set of cardinality n. There is an exact algorithm that computes PCOD(A,z) in O(kn) time and O(kn) memory.*

2.5.2 Arbitrary Target Sets

We shall try to adapt the techniques used in the previous section for arbitrary target sets, too. Let A be a finite set of strings and $z \in CO_*(A)$; denote by

$$MaxPrefLen(A,z) = \begin{cases} |z|, \text{ iff } z \in A, \\ \\ \max\{q \mid q < |z|, \text{ there exists } x \in A, \\ |x| > q, \text{ so that } x[1,q] = z[1,q]\}, \end{cases}$$

$$\begin{aligned} &MaxSufLen(A,z) = \max\{q \mid \text{there exists } x \in A, \\ &|x| = |z|, \text{ so that } x[|x|-q+1,|x|] = z[|z|-q+1,|z|]\}, \\ &ArbMaxSubLen(A,z,p) = \max\{q \mid \text{there exists } x \in A \text{ and} \\ &|x| \geq p+q \text{ such that } x[p,p+q-1] = z[p,p+q-1]. \end{aligned}$$

We define iteratively the set $ArbH(A,z)$ of intervals of natural numbers as follows:

1. $ArbH(A,z) = \{[1, MaxPrefLen(A,z)]\}$;
2. Take the interval $[i,j]$ having the largest j; if $j = |z|$, then stop. If $j < |z| - MaxSufLen(A,z)$, then put the new interval $[j+1, j+ArbMaxSubLen(A,z,j+1)]$ into $ArbH(A,z)$; otherwise put $[j+1,|z|]$ into $ArbH(A,z)$.

Theorem 2.5.4.

1. Let A be a finite set of strings and B be a finite subset of $CO_*(A)$. Then

$$\frac{\sum_{z\in B}(card(ArbH(A,z))-1)}{2} \leq SCOD(A,B) \leq$$
$$\sum_{z\in B}(card(ArbH(A,z))-1).$$

2. There exist A and $B \subseteq CO_*(A)$ such that

$$SCOD(A,B) = \frac{\sum_{z\in B}(card(ArbH(A,z))-1)}{2}.$$

3. There exist A and $B \subseteq CO_*(A)$ such that

$$SCOD(A,B) = \sum_{z\in B}(card(ArbH(A,z))-1).$$

Proof. 1. We shall prove the first assertion by induction on the length of the longest string in B, say k. The non-trivial relation is

$$\frac{\sum_{z\in B}(card(ArbH(A,z))-1)}{2} \leq SCOD(A,B). \qquad (*)$$

If $k = 1$, then $B \subseteq A$, hence $card(H(A,z)) = 1$ for all $z \in B$, therefore the relation $(*)$ is satisfied. Assume that the relation $(*)$ holds for any two finite sets X and Y, $Y \subseteq CO_*(X)$, all strings in Y being shorter than k. Assume that $B \setminus A = \{z_1, z_2, \ldots, z_m\}$ and let $S = s_1, s_2, \ldots, s_q$, $s_i = (x_i, y_i) \vdash_{p_i} (u_i, v_i)$, $1 \leq i \leq q$, be a $B \setminus A$-producing CCS in $CO_*(A)$. Note that at least one string in $B \setminus A$ should exist, otherwise the relation $(*)$ being trivially fulfiled.

Consider m new symbols $a_1, a_2, \ldots, a_m$ and construct the sets

$$\begin{aligned} A' &= \{x[1,r]a_ix[r+2,|x|] | x \in A, 1 \leq i \leq m\}, \\ B' &= \{z_i[1,r]a_iz_i[r+2,|z_i|] | 1 \leq i \leq m\}, \end{aligned}$$

where $r = \min\{p_i | 1 \leq i \leq q\}$. One can construct a B'-producing CCS in $CO_*(A')$ of the same length of S, say S' by applying the next procedure.

Algorithm 2.5.3.

```
Procedure Convert(S);
begin
for $j := 1$ to $m$ begin
   $z := z_j$; $t := m$;
   while $z \notin A$ begin
      find the maximal $l \leq t$ such that $u_l = z$ or $v_l = z$;
      $t := l - 1$;
      if $u_l = z$ then replace $u_l$ by $u_l[1, r]a_j u_l[r + 2, |u_l|]$;
         if $p_l > r$ then $z := x_l$;
            replace $x_l$ by $x_l[1, r]a_j x_l[r + 2, |x_l|]$;
         else $z := y_l$;
            replace $y_l$ by $y_l[1, r]a_j y_l[r + 2, |y_l|]$;
         endif;
      else replace $v_l$ by $v_l[1, r]a_j v_l[r + 2, |v_l|]$;
         if $p_l \leq r$ then $z := x_l$;
            replace $x_l$ by $x_l[1, r]a_j x_l[r + 2, |x_l|]$;
         else $z := y_l$;
            replace $y_l$ by $y_l[1, r]a_j y_l[r + 2, |y_l|]$;
         endif;
      endif;
   endwhile;
   replace $z$ by $z[1, r]a_j z[r + 2, |z|]$;
endfor;
replace the symbol on the position $r + 1$ in all strings in $S$ that have not been replaced so far by $a_1$;
end.
```

For a better understanding of the previous procedure we provide below the effect of applying this procedure to a B-producing CCS, $B = \{abacdb, aabccb, bbaadc\}$, starting from the initial set $A = \{abbccb, aaaadb, bbbcdc\}$. The CCS S is

$$
\begin{aligned}
&(abbccb, aaaadb) \vdash_2 (abaadb, aabccb),\\
&(abbccb, abaadb) \vdash_3 (abbadb, abaccb),\\
&(bbbcdc, abaccb) \vdash_2 (bbaccb, abbcdc),\\
&(bbaccb, aaaadb) \vdash_3 (bbaadb, aaaccb),\\
&(bbaadb, bbbcdc) \vdash_5 (bbaadc, bbbcdb),\\
&(abaadb, aaaccb) \vdash_2 (abaccb, aaaadb),
\end{aligned}
$$

$$(abaccb, aaaadb) \vdash_4 (abacdb, aaaacb).$$

The procedure *Convert* runs for $r = 2$ transforming this sequence into the sequence S':

$$\begin{aligned}
&(aba_2ccb, aaa_3adb) \vdash_2 (aba_3adb, aaa_2ccb),\\
&(aba_1ccb, aba_3adb) \vdash_3 (aba_1adb, aba_3ccb),\\
&(bba_1cdc, aba_3ccb) \vdash_2 (bba_3ccb, aba_1cdc),\\
&(bba_3ccb, aaa_1adb) \vdash_3 (bba_3adb, aaa_1ccb),\\
&(bba_3adb, bba_1cdc) \vdash_5 (bba_3adc, bba_1cdb),\\
&(aba_1adb, aaa_1ccb) \vdash_2 (aba_1ccb, aaa_1adb),\\
&(aba_1ccb, aaa_1adb) \vdash_4 (aba_1cdb, aaa_1acb).
\end{aligned}$$

Now we apply Algorithm 1 to the sequence S' for r previously defined. The obtained sequence S'' is a B''-producing CCS in $CO_*(A'')$, where

$$\begin{aligned}
A'' &= \{a_i x[r+2, |x|] | x \in A, 1 \leq i \leq m\},\\
B'' &= \{a_i z_i[r+2, |z_i|] | 1 \leq i \leq m\}
\end{aligned}$$

due to the two claims from the proof of Lemma 2.

For each $1 \leq i \leq m$ $card(ArbH(A'', a_i z_i[r+2, |z_i|]))$ is either $card(ArbH(A, z_i))$ or $card(ArbH(A, z_i)) - 1$. Furthermore, for each i such that $card(ArbH(A'', a_i z_i[r+2, |z_i|])) = card(ArbH(A, z_i)) - 1$ there exist at least one step in S' where the strings exchange prefixes of length at most r. It follows that $lg(S'') \leq lg(S') - \lceil t/2 \rceil$, where $t = card(\{i | card(ArbH(A'', a_i z_i[r+2, |z_i|])) = card(ArbH(A, z_i)) - 1\})$. Consequently,

$$\begin{aligned}
lg(S) &= lg(S') \geq lg(S'') + \lceil t/2 \rceil \geq\\
&\frac{\sum_1^m (card(ArbH(A'', a_i z_i[r+2, |z_i|])) - 1)}{2} + \lceil t/2 \rceil \geq\\
&\frac{\sum_1^m (Arbcard(H(A, z_i)) - 1)}{2}.
\end{aligned}$$

The reader may easily find sets A and B fulfilling the last two assertions. □

An α-approximation algorithm for a minimization problem is a polynomial algorithm that delivers a solution whose value is at most α times the minimum. From the previous theorem we have:

Theorem 2.5.5. *There is an 2-approximation algorithm for computing the sequential crossover distance from two sets of strings.*

Proof. It is easy to notice that an algorithm that computes $\sum_{z \in B}(card(ArbH(A,z)) - 1)$ requires $O(n|B|)$, where $n = card(A)$ and $|B|$ is the sum of the lengths of all strings in B. □

As far as the parallel crossover distance is concerned one may state

Theorem 2.5.6. *There is an exact algorithm that computes $PCOD(A,B)$ in $O(n|B|)$ time, where $n = card(A)$ and $|B|$ is the sum of the lengths of all strings in B.*

We have introduced two crossover distances between two finite sets of strings and proposed some algorithms for computing them based on the "greedy" strategy. All results presented here are valid for a particular type of crossover, namely the uniform crossover where the strings exchange with each other prefixes of the same length. Even so, the problem of finding a polynomial algorithm to compute the sequential crossover between two finite sets remains open. The next step is naturally to consider the case of arbitrary crossover; we hope to return to this in a forthcoming paper.

Chapter 3

Language Generating Devices

Generative devices based on operations suggested by the mutations which take place within genomes appear very attractive for formal language theorists (see [33, 34, 37, 89, 94]) and hopefully, biologists (see [11, 19, 20, 53, 116, 121, 122, 123, 134]). It seems that an increasing trend manifests itself throughout the field of computational biology toward abstracted views of biological sequences, which is very much in the spirit of language theory.

The issues addressed by this section are basically formal language theoretic questions but the results presented here address a biologically important and realistic problem as well. We hope that our model responds to some computational aspects of bioinformatics. Maybe it is worth mentioning here that, in spite of the simplicity of our model (no context constraints, no auxiliary symbols), the decidability status of many important problems is negative.

It might also be argued that, due to practical problems when dealing with arbitrarily large genomes, the length of strings is of a definite importance. Thus, following the approach in L-systems area, it appears of interest to study length sets or growth functions associated to the evolutionary grammars. One paragraph of this section is a first approach in this respect.

We introduce in this first section the most general model - the evolutionary grammar - which takes into account not only all the operations involved in the genome evolution but lexical context for controlling their application. We introduce a grammatical model for the evolution of genomes on the basis of gene mutations and chromosome mutations and present some properties of such grammars. Few problems which might be biologically relevant are discussed from the computational point of view. On the other hand, the model suggests a new direction in formal language theory motivated by the common operations of genome evolution.

3.1 Evolutionary Grammars

For an alphabet V we denote by $C(V) = \{(w) | w \in V^+\}$, $(,) \notin V$.

The *C-length* of $x \in C(V)^+$ is defined as follows:

$$lg_C(x) = \begin{cases} 1, \text{ if } x \in C(V) \\ lg_C(y) + 1, \text{ if } x = yw, w \in C(V) \end{cases}$$

An *evolutionary grammar* is a construct

$$EG = (V, GM, CE, CO, A)$$

where

- V is an alphabet (the set of nucleotides).
- $GM \subseteq \{Sub, Ins, Del\}$ (the set of gene mutations: substitutions, insertions, deletions, respectively)
  - Sub is a subset of $V \times (V \cup \{(\}) \times (V \cup \{)\}) \times V$
  - Ins, Del are subsets of $V \times (V \cup \{(\}) \times (V \cup \{)\}) \setminus (V \times \{(\} \times \{)\})$
- $CE \subseteq \{CDel, Inv, Trans, Dupl\}$ (the set of chromosomes evolutions: deletions, inversions, transpositions, duplications, respectively)
  - $CDel$ and Inv are finite subsets of $C(V)^+$
  - $Trans$ and $Dupl$ are finite subsets of $(C(V)^+)^3$

- CO is a finite subset of $(C(V)^+)^4$ such that if $(x, y, z, t) \in CO$, then $(z, t, x, y) \in CO$, too (the set of crossing-over operations).
- A is a finite subset of $C(V)^+$ (the set of initial genomes)

We define the following relations on the set of genomes $C(V)^+$:

(i) $x \Longrightarrow_{GM} y$ iff one of the following conditions holds:
1. $x = uacbv,\ y = uadbv,\ (c, a, b, d) \in Sub \in GM,$
2. $x = uabv,\ y = uacbv,\ (c, a, b) \in Ins \in GM,$
3. $x = uacbv,\ y = uabv,\ (c, a, b) \in Del \in GM$

(these rules model the gene mutations)

(ii) $x \Longrightarrow_{CE} y$ iff one of the following conditions holds:
1. $x = x_1x_2x_3,\ y = x_1x_3,\ x_2 \in CDel \in CE,$
2. $x = x_1x_2x_3,\ y = x_1mi(x_2)x_3,\ x_2 \in Inv \in CE,$
3. $x = x_1x_2x_3x_4x_5x_6,$

$$y = \begin{cases} x_1x_3x_4x_2x_5x_6, & (x_2, x_4, x_5) \in Trans \in CE \\ x_1x_2x_5x_3x_4x_6, & (x_5, x_2, x_3) \in Trans \in CE \\ x_1x_2x_3x_4x_2x_5x_6, & (x_2, x_4, x_5) \in Dupl \in CE \\ x_1x_2x_5x_3x_4x_5x_6, & (x_5, x_2, x_3) \in Dupl \in CE \end{cases}$$

(these rules model the chromosomes rearrangements)

(iii) $x, y \Longrightarrow_{CO} z$ iff $x = x_1x_2x_3x_4,\ y = y_1y_2y_3y_4$, and
1. $lg_C(x_1x_2) = lg_C(y_1y_2),$
2. $z = x_1x_2y_3y_4$

(this rule models the crossing-over of the genomes x and y resulting in z).

Let us define the following sequences of languages

$$\begin{aligned} L_0(EG) &= A, \\ L_{i+1}(X) &= \{y \in C(V)^+ \mid x \Longrightarrow_X y \text{ for some } x \in L_i(EG)\}, \\ &\quad X \in \{GM, CE\},\ i \geq 0, \\ L_{i+1}(CO) &= \{z \in C(V)^+ \mid x, y \Longrightarrow_{CO} z \text{ for some} \\ &\quad x, y \in L_i(EG)\},\ i \geq 0, \end{aligned}$$

$$\begin{aligned} L_{i+1}(EG) &= L_i(EG) \cup L_{i+1}(GM) \cup L_{i+1}(CE) \cup \\ &\quad L_{i+1}(CO),\ i \geq 0. \end{aligned}$$

The world generated by an evolutionary grammar as above is

$$W(EG) = lim_{n \longrightarrow \infty} L_n(EG)$$

(intuitively, $L_i(EG)$ contains all genomes which can be obtained from genomes in the set A after at most i mutations and $W(EG)$ is the union of all these sets, i.e. it consists of all genomes which originate from elements of A by some given mutations).

As one can easily see the aforementioned definition of the evolutionary grammars tries to model all local as well as global operations that might occur during the evolution time.

We say that an evolutionary grammar EG is *local* or *global* if $CE = \emptyset$ or $GM = \emptyset$ holds, respectively. Moreover, EG is called *non-deleting*, if $Del = CDel = \emptyset$ holds.

The following matters appear to be of interest from the computational biology point of view:

1. It is possible to get a given genome from another one ?

2. Is a world generated by a given evolutionary grammar finite or infinite ?

3. Are there common genomes in two given worlds ?

4. What can be said on the number of genoms derivable from a given set of genoms by a certain number of given mutations ?

5. What can be said on the length of genoms derivable from a given set of genoms by a certain number of given mutations ?

We mention that our model is not satisfactory in order to describe the process of evolution because we take into consideration all genomes created by the given mutations whereas the nature takes only some of them which survive since the corresponding organisms have better properties and abilities (the others lead to lethal situations). In order to model this aspect one has to add further features (see [28] for an approach).

Furthermore, we give the mutations in the grammar and allow only them during the evolution. However, which mutations lead to new organism is not known in advance. Therefore one has to add a mechanism which selects the mutations and does not require their knowledge in advance. We shall present answers to some of the problems mentioned above.

3.1.1 Decision Problems

In this subsection we are firstly interested in the question whether or not a given genom can be transformed by some given mutations to a given genom. Formally, this can be written as follows: Given some sets V of nucleotides, $GM = \{Sub, Ins, Del\}$, $CE = \{CDel, Inv, Trans, Dupl\}$ and CO of mutations and genomes x and y, does there exist a derivation

$$x \Longrightarrow_{X_1} x_1 \Longrightarrow_{X_2} x_2 \Longrightarrow_{X_3} \cdots \Longrightarrow_{X_{n-1}} x_{n-1} \Longrightarrow_{X_n} y$$

with $n \geq 1$ and $X_i \in \{GM, CE, CO\}$ for $1 \leq i \leq n$. Obviously, this is equivalent to the following problem: Given an evolutionary grammar $EG = (V, GM, CE, CO, \{x\})$ and a genom y, does $y \in W(EG)$ hold. This is the membership problem which is well-known and well investigated in the theory of formal languages (see [64]). However, the operations which are performed in one derivation step of a evolutionary grammar differ essentially from the replacements used in the classical theory of formal languages.

Theorem 3.1.1 *i) There is no algorithm which decides, for a given local or global evolutionary grammar EG (with a singleton set of initial genomes) and a given genome y, whether or not $y \in W(EG)$.*

ii) There is an algorithm which decides, for a given non-deleting evolutionary grammar EG and a given genome y, whether or not $y \in W(EG)$.

Proof. i) It is well-known that there is no algorithm which decides the membership problem for arbitrary phrase structure grammar, i.e. which decides, for a grammar $G = (N, T, P, S)$ and $w \in T^+$, whether or not $w \in Gen(G)$ (see [64]).

Let G be a phrase structure grammar as above, whose set of productions P contains only rules of the following forms:

$$AB \rightarrow AC, \qquad AB \rightarrow CB$$

$$\begin{aligned} A &\rightarrow BC \\ A &\rightarrow \varepsilon, \qquad A \rightarrow a \end{aligned}$$

with $A, B, C \in N$ and $a \in T$. For the effect of a rule $AB \rightarrow CD$ could be obtained without side effects by the context sensitive rules $AB \rightarrow YB$, $YB \rightarrow YX$, $YX \rightarrow CX$, $CX \rightarrow CD$, provided that Y and X have no other occurences in the rules of G and by the Kuroda normal form, we claim that the aforementioned forms do not induce any restriction of the generative capacity.

Take the local evolutionary grammar

$$EG = (V, \{Sub, Ins, Del\}, \emptyset, \emptyset, \{(S)\})$$

where

$$\begin{aligned} V &= N \cup T \cup \{[B], [BC] | A \rightarrow BC \in P\} \\ Sub &= \{(B, A, X, C) | AB \rightarrow AC \in P, X \in N \cup T \cup \{)\}\} \\ &\cup \{(A, X, B, C) | AB \rightarrow CB \in P, X \in N \cup T \cup \{(\}\} \\ &\cup \{(A, X, Y, a) | A \rightarrow a \in P, X \in N \cup T \cup \{(\}, \\ &\quad Y \in N \cup T \cup \{)\}\} \\ &\cup \{(A, X, Y, [BC]) | A \rightarrow BC \in P, X \in N \cup T \cup \{(\}, \\ &\quad Y \in N \cup T \cup \{)\}\} \\ &\cup \{([BC], [B], X, C) | B, C \in N, X \in N \cup T \cup \{(\}\} \\ &\cup \{([B], X, Y, B) | B \in N, X \in N \cup T \cup \{(\}, Y \in N \\ Ins &= \{([B], X, [BC] | B, C \in N, X \in N \cup T \cup \{(\}\} \\ Del &= \{(A, X, Y) | A \rightarrow \varepsilon \in P, X \in N \cup T \cup \{(\}, Y \in N \cup T \cup \{)\}\} \end{aligned}$$

Clearly, $w \in L$ iff $(w) \in W(EG)$ that implies the undecidability of the membership problem for local evolutionary grammars.

For the same phrase structure grammar G let us assume that $P = \{\alpha_i \rightarrow \beta_i | 1 \leq i \leq n$, for some $n \geq 1$. We consider the global evolutionary grammar

$$EG = (V, \emptyset, \{CDel, Inv, Dupl\}, \emptyset, \{x\})$$

where

$$V = N \cup T \cup \{c_1, c_2, d_1, d_2, \#, \$\} \text{ with}$$

$$\begin{aligned}
& \{c_1, c_2, d_1, d_2, \#, \$\} \cap (N \cup T) = \emptyset \\
x &= g(c_1c_2d_1d_2\beta_1d_2\beta_2 \ldots d_2\beta_n)(\#)(S)(\$) \\
CDel &= \{g(d_1\alpha_i d_1d_2) | 1 \leq i \leq n\} \cup \{(c_1), (c_2)\} \\
Inv &= \{(c_1)(X) | X \in N \cup T\} \cup \{(X)(c_2) | X \in N \cup T\} \\
Dupl &= \{(g(d_2\beta_i), g(d_1\alpha_i d_1), (X)) | 1 \leq i \leq n, X \in N \cup T \cup \{\$\}\} \cup \\
& \{((c_1), (\#), (X)) | X \in N \cup T\} \cup \{((c_2), (X), (\$)) | \\
& X \in N \cup T\} \cup \{((d_1), (c_1), (X)) | X \in N \cup T\} \cup \\
& \{((d_1), (X), (c_2)) | X \in N \cup T\}
\end{aligned}$$

In the above relations g is a morphism from $N \cup T$ into $N \cup T \cup \{(,)\}$ defined as $g(X) = (X)$, for all $X \in N \cup T$.

Thus, by deletions and duplications we are able to simulate all productions of P. Therefore,

$$w \in L \text{ iff } g(c_1c_2d_1d_2\beta_1d_2\beta_2 \ldots d_2\beta_n)(\#)g(w)(\$) \in W(EG)$$

which concludes the first statement of this theorem.

ii) Since the grammar is non-deleting, any step in the derivation does not decrease the length of the generated word. Thus one can construct an upper bound n for the number of steps which are necessary in order to obtain a given element y by a given grammar EG. Now we only have to determine in succession all sets $L_0(EG)$, $L_1(EG)$, ..., $L_n(EG)$ which can be done algorithmically since all these sets and the sets of operations are finite. Finally, we have to check whether or not $y \in L_n(EG)$. □

The next decidability results are direct consequences of the previous theorem.

Corollary 3.1.1 *i) There is no algorithm which decides, for two given local/global evolutionary grammars EG_1 and EG_2, whether or not $W(EG_1) \subseteq W(EG_2)$.*

ii) There is no algorithm which decides, for two given local/global evolutionary grammars EG_1 and EG_2, whether or not $W(EG_1) \cap W(EG_2) = \emptyset$.

iii) There is no algorithm which decides, for two given local/global non-deleting evolutionary grammars EG_1 and EG_2, whether or not $W(EG_1) \cap W(EG_2) = \emptyset$.

Proof. In order to prove the first assertion it suffices to take an evolutionary grammar EG_1 generating only one genome and another arbitrary one EG_2 generating a nonrecursive world. An algorithm for solving the problem $W(EG_1) \subseteq W(EG_2)$ would imply that $W(EG_2)$ is recursive, contradiction.

The undecidability of the intersection emptyness problem for local (non-deleting) evolutionary grammars follows obviously from Theorem 3.1.1. Let $G_i = (N_i, T_i, S_i, P_i)$, $i = 1, 2$, be two arbitrary grammars with $N_1 \cap N_2 = \emptyset$ and $P_i = \{\alpha^i_j \to \beta^i_j | 1 \leq j \leq n_i\}$, $i = 1, 2$, for some $n_1, n_2 \geq 1$. As in the proof of Theorem 3.1.1 one can construct two global evolutionary grammars EG_1 and EG_2 such that $W(EG_1) \cap W(EG_2)$ is the following set

$$g(c_1c_2d_1d_2\beta^1_1d_2\beta^1_2 \ldots d_2\beta^1_nd_2\beta^2_1d_2\beta^2_2 \ldots d_2\beta^2_n)(\#)$$
$$g(Gen(G_1) \cap Gen(G_2))(\$).$$

Of course, $W(EG_1) \cap W(EG_2) = \emptyset$ iff $Gen(G_1) \cap Gen(G_2) = \emptyset$, which is undecidable.

But the above construction does not work for global non-deleting evolutionary grammars. So, we shall provide below another proof in the case of non-deleting evolutionary grammars.

Let

$$x = (x_1, x_2, \ldots, x_n) \quad \text{and } y = (y_1, y_2, \ldots, y_n)$$

be an instance of the Post Correspondence Problem over the alphabet $\{a, b\}$. Consider the global non-deleting evolutionary grammars EG_x, EG_y as follows:

$$EG_z = (V, \emptyset, \{Inv_z, Dupl_z\}, \emptyset, A_z),\ z \in \{x, y\},$$

where

$$\begin{aligned} V &= \{1, 2, \ldots, n\} \cup \{a, b, c, d\}, \\ A_x &= \{g(mi(x_1)1mi(x_2)2 \ldots mi(x_n)nmi(y_1)1mi(y_2)2 \ldots \\ &\quad mi(y_n)nkcdx_k) | 1 \leq k \leq n\} \\ A_y &= \{g(mi(x_1)1mi(x_2)2 \ldots mi(x_n)nmi(y_1)1mi(y_2)2 \ldots \end{aligned}$$

$$\begin{aligned}
& mi(y_n)nkcdy_k)|1 \leq k \leq n\} \\
Dupl_x &= \{(g(mi(x_i)i), (p)(q), (c)(d))|1 \leq i, p, q \leq n\} \\
Dupl_y &= \{(g(mi(y_i)i), (p)(q), (c)(d))|1 \leq i, p, q \leq n\} \\
Inv_x &= \{g(mi(x_i)icd)|1 \leq i \leq n\} \cup \{(d)(c)(i)|1 \leq i \leq n\} \\
Inv_y &= \{g(mi(y_i)icd)|1 \leq i \leq n\} \cup \{(d)(c)(i)|1 \leq i \leq n\}
\end{aligned}$$

Note that g is the same morphism as that from the proof of Theorem 3.1.1. One easily observe that $W(EG_x) \cap W(EG_y) = \emptyset$ if and only if the given instance has no solution and the proof is complete. □

Theorem 3.1.2 *It is undecidable whether or not a given local/global evolutionary grammar generates a finite number of genoms.*

Proof. By applying a reduction to the Post Correspondence Problem one can prove (see [96]) that the finiteness problem for the sentential form languages of context-sensitive grammars is undecidable. Following the proof of Theorem 3.1.1 we get a local/global evolutionary EG grammar simulating all productions of a given context sensitive grammar G. By that simulation every sentential form of G has a finite number of "similar" genomes in $W(EG)$ (since G is length increasing) and each genome in $W(EG)$ is associated to a sentential form of G. Therefore, $W(EG)$ is finite if and only if the sentential form language of G is finite. □

It is worth mentioning here that the problem remains undecidable for local non-deleting evolutionary grammars as well. We do not know whether the problem has the same decidability status for global non-deleting evolutionary grammars.

All the results above are negativist. For lightening a bit this sombre vision we present the next decidable result.

Theorem 3.1.3 *It is decidable whether or not a given global, non-deleting evolutionary grammar produces all genomes consisting from chromosomes in a given finite set.*

Proof. The problem can be stated as follows. Let $EG = (V, \emptyset, CE \setminus \{CDel\}, CO, A)$ be an evolutionary grammar and F be a given finite subset of $C(V)$. Now, we have to decide whether or not $W(EG) = F^+$ holds. Take $k = max\{lg_C(x)|x \in A\}$. Our problem can be reduced to the checking of the following relations:

(i) $F = \{w \in C(V)|$ exist $x, y \in C(V)^*$ such that $xwy \in A$

(ii) Any w with $lg_C(w) \leq k+1$ belongs to $W(EG)$

Note that the relations above can be algorithmically verified since the evolutionary grammar is non-deleting. If one of the above relations is not fulfilled, then the equality $W(EG) = F^+$ does not hold, too. Now it suffices to show how can be produced any genome $w \in F^+$ with $lg_C(w) = k+2$. Let $wx \in F^+$ with $lg_C(w) = k+1$, $lg_C(x) = 1$. As $lg_C(w) = k+1$ there exists $y \in A$ with $lg_C(y) \leq k$ such that w can be obtained from y by different chromosome mutations. On the other hand, the genome yx has its C-length at most $k+1$; consequently it is produced from a genome $z \in A$. Therefore, z can produce wx as well which concludes the proof. □

It is worth mentioning here that the theorem remains valid when F is a finite subset of $C(V)^+$.

3.1.2 A Growth Function for Genomes

Based on the definition of the chromosome length lg_C, we define the *growth function* $f_{EG} : \mathrm{N} \to \mathrm{N}$ of an evolutionary grammar EG by

$$f_{EG}(n) = \max\{lg_C(x) \mid x \in L_n(EG)\}.$$

This function is an analogon of the growth function known from the theory of LINDENMAYER systems (see [113]).

First we mention that the gene mutations GM do not influence the growth function because they change the length of genes but do not change the length of a chromosome. Because $L_i(EG) \subseteq L_{i+1}(EG)$, f_{EG} is a monotonous function.

Let

$$a = \max\{cl(x) \mid x \in A\}.$$

If the set $Dupl$ of duplications is empty, then no increase of the chromosome length is possible, i.e.

$$f_{EG}(n) = a$$

is a constant function in this case.

On the other hand, if $k = \max\{cl(x_1) \mid (x_1, x_2, x_3) \in Dupl\}$, then any derivation step can increase the C-length at most by k. Thus we obtain

$$f_{EG}(n) \leq a + n \cdot k.$$

This shows that, for any evolutionary grammar EG, the growth function is bounded from below by an constant function and bounded from above by a linear function.

However, a lot of other mappings within the aforementioned bounds may be growth functions of some evolutionary grammars. We provide now such a class of mappings. A function $f : \mathbf{N} \to \mathbf{N}$ is *monotonously increasing ultimately periodically linear* (MIUPL, shortly) if there are numbers $t \geq 0$ (*threshold*), $p \geq 1$ (*period*) and $0 \leq r_1 \leq r_2 \leq \ldots \leq r_p$ satisfying the condition

$$f(t + ip + j) = f(t) + ir_p + r_j, \quad \text{for all } i \geq 0,\ 1 \leq j \leq p$$

We give the following result on the existence of growth functions.

Theorem 3.1.4 *i) Every MIUPL mapping is the growth function of an evolutionary grammar.*

ii) There is an evolutionary grammar EG such that $f_{EG}(n) = O(\sqrt{n})$.

Proof. The first item is left to the reader and we only prove ii).

It is sufficient to consider the global evolutionary grammar

$$EG = (V, \emptyset, \{Inv, Trans, Dupl\}, \emptyset, \{eabcdaaf\})$$

with

$$\begin{aligned} Inv &= \{ac, bc, de\}, \\ Trans &= \{(aec, a, f), (f, a, e)\}, \\ Dupl &= \{(bc, a, a), (a, c, e), (e, bc, cd)\} \end{aligned}$$

where a, b, c, d, e, f are the elements of $C(V)$ used. First we obtain the following derivation

$$\begin{aligned} eabcdaaf \implies\ & eabcdabcaf \implies eabcdacbaf \implies eabcdcabaf \implies \\ & eabccdabaf \implies eabcecdabaf \implies eabcaecdabaf \implies \\ & eabcdabaaecf \end{aligned}$$

which is unique in that sense that the application of other mutations do not increase the C-length and lead to situations such that no further applications of mutations are possible). Now we can repeat this derivation with an additional application of the transposition (f, a, e) and two additional inversions and obtain after 10 steps *eabcdababaaecfec* (again, this derivation is unique in the above sense; however, the time where the additional transposition is performed is not uniquely determined). Now we can repeat this process and have to use two additional steps in order to obtain *eabcdabababaaecfecec* etc. Therefore we get

$$f_{EG}(0) = 8,\ f_{EG}(7) = 12,\ f_{EG}(17) = 16,\ f_{EG}(29) = 20$$

and generally

$$f_{EG}(7 + 8n + 2\sum_{i=1}^{n} i) = f_{EG}(n^2 + 9n + 7) = 12 + 4n$$

which proves the statement. □

In light of the last result we suspect that the class of evolutionary grammar growth functions is likely quite large and it seems to be very difficult to give an exhaustive characterization.

3.2 Context-Free Evolutionary Grammars

This section is devoted to the context-free variant of the evolutionary grammars investigated in the previous section. For sake of simplicity, in the rest of this section we refer to these grammars as evolutionary grammars but the reader should understand that they are context-free evolutionary grammars. Besides all the problems studied for the evolutionary grammars in the previous section we shall consider here some specific problems as well.

A *context-free evolutionary grammar* ([33]) is a construct

$$EG = (V, A, Del, Inv, Xpos, Dup)$$

where

- V is an alphabet (the set of nucleotides. Strings of V^* are referred as genomes).
- Del, Inv, $Xpos$ and Dup are finite subsets of V^+ (the sets of deletions, inversions, transpositions and duplications, respectively),
- A is a finite subset of V^+ (the set of initial genomes)

We define the following relationships on the set V^+:

$x \Longrightarrow_{Del} y$ iff $x = uvw,\ y = uw,\ v \in Del$

x evolves into y by deleting a segment

$x \Longrightarrow_{Inv} y$ iff $y \in \mathcal{I}_{(V,Inv)}(x)$

x evolves into y by reversing the orientation of a segment

$x \Longrightarrow_{Xpos} y$ iff $y \in \mathcal{T}_{(V,Xpos)}(x)$

x evolves into y by transposing a segment to a new position

$x \Longrightarrow_{Dup} y$ iff $y \in \mathcal{D}_{(V,Dup)}(x)$

x evolves into y by copying a segment

and

$$x \Longrightarrow_{EG} y \text{ iff } x \Longrightarrow_X y, \text{ for some } X \in \{Del, Inv, Xpos, Dup\}$$

Denote by $\Longrightarrow_X^*$ the reflexive and transitive closure of $\Longrightarrow_X$. The language generated by an evolutionary grammar as above is

$$L(EG) = \{w \in V^* | x \Longrightarrow_{EG}^* w, \text{ for some } x \in A\}$$

(intuitively, $L(EG)$ consists of all genomes which originate from elements of A by some given mutations).

Remark. Each evolutionary grammar may be viewed as a very particular pure grammar [47]. Indeed, each mutation can be simulated by a set of pure productions as follows.

- For each $x \in Del$, the associated production is $x \longrightarrow \varepsilon$.
- For each $x \in Inv$, the associated production is $x \longrightarrow mi(x)$.

- For each $x \in Xpos$, the associated productions are $xa \longrightarrow ax$, for all $a \in V$.
- For each $x \in Dup$, the associated production is $x \longrightarrow xx$.

An evolutionary grammar is called *non-deleting* if $Del = \emptyset$.

We shall give an informal biological interpretation of our generative device. The alphabet of the grammar might be considered as the alphabet of nucleotides and the set A as the set of initial genomes. Evolutionary events are described by the sets Del, Inv, $Xpos$, Dup; thus the language generated by an evolutionary grammar may be viewed as the world consisting of all genomes which originate from elements of A by some given mutations.

3.2.1 Computational Power

Denote by $\mathcal{L}(EG)$ the family of languages generated by evolutionary grammars.

Theorem 3.2.1 *1. The family of languages generated by evolutionary grammars is incomparable with the family of regular languages.*

2. The family of languages generated by evolutionary grammars is incomparable with the family of context-free languages.

Proof. The following evolutionary grammar generates a non-context-free language:

$$EG = (\{a,b,c\}, \{abc\}, \emptyset, \emptyset, \{a,b,c\}, \{abc\})$$

It is easy to check that

$$L(EG) = \{x \in \{a,b,c\}^+ \mid |x|_a = |x|_b = |x|_c\}$$

Let us consider the alphabet $V = \{a,b,c,d,e\}$ and the language

$$L = \{x \in V^* \mid \text{if } x = x_1\alpha_1\alpha_2x_2, \text{ with } x_1,x_2 \in V^*, \text{ and } \alpha_1,\alpha_2 \in V, \\ \text{then } \alpha_1 \neq \alpha_2 \text{ and } \alpha_1\alpha_2 \neq ba\}.$$

It is easy to see that this is an infinite regular language. For instance, L contains all square-free strings over $\{c,d,e\}$, and one knows that this language is infinite (see [128], [118]). The regularity of L can be easily checked. Moreover, if $Sub(L)$ is the language of all substrings of strings in L, then obviously $L = Sub(L)$ (the properties of strings in L are hereditary).

Let us assume that L is generated by an evolutionary grammar, i.e. $L = L(G)$ for some $G = (V, A, Del, Inv, Xpos, Dup)$.

(i) Assume that there is $z \in Del, z \neq \varepsilon$, such that $z \in Sub(L)$. This means that $z \in L$. Suppose that $z = \alpha_1 z_1 = z_2 \alpha_2$, for some $z_1, z_2 \in V^*$ and $\alpha_1, \alpha_2 \in V$. There are strings $w = w_1 \beta z \beta w_2$ in L, with $w_1, w_2 \in V^*, \beta \notin \{a, b, \alpha_1, \alpha_2\}$ (we have $card(V) = 5$ and $z \in L$, hence at least strings of this form with $w_1 = w_2 = \varepsilon$ can be found in L). Then $w \Longrightarrow_{Del} w_1 \beta\beta w_2$; the obtained string is not in L, a contradiction.

(ii) Assume that there is $z \in Xpos, z \neq \varepsilon$, such that $z \in Sub(L)$. For $z = \alpha_1 z_1 = z_2 \alpha_2$, for some $z_1, z_2 \in V^*$ and $\alpha_1, \alpha_2 \in V$, we take $w = w_1 \beta z \beta w_2$ in L, with $w_1, w_2 \in V^*, \beta \notin \{a, b, \alpha_1, \alpha_2\}$, and we get a contradiction by noticing that $w \Longrightarrow_{Xpos} w_1 \beta\beta z w_2$ produces a string not in L.

Therefore, no element of Del and $Xpos$ can be used in a derivation step with respect to G, we can replace these sets by $\emptyset$ and the generated language is the same. Thus, we suppose that G has already these components empty.

(iii) Consider now the language $SF_{\{c,d,e\}}$, of all square-free strings over $\{c, d, e\}$. We have mentioned above that this is an infinite language ([128], [118]). Construct the strings of the form

$$\alpha_1 ab \alpha_2 ab \ldots \alpha_k ab \alpha_{k+1},$$

for $k \geq 1, \alpha_i \in \{c, d, e\}, 1 \leq i \leq k+1$, and $\alpha_1 \alpha_2 \ldots \alpha_{k+1} \in SF_{\{c,d,e\}}$. Denote by M the language of such strings.

The language M is infinite (because $SF_{\{c,d,e\}}$ is infinite) and it is included in L (no double symbol and no substring ba appear in its strings). The axiom set A is finite. Therefore, there are strings $w \in M$ which are not in A, that is there is a derivation $x \Longrightarrow^* w$ in G. Let $w' \Longrightarrow w$ be the last step of such a derivation. Without loss of generality we may assume that $w' \neq w$ (we ignore the steps which do not observe this property). By the definition, we have $w' \in L$.

Because w contains no square, the derivation step $w' \Longrightarrow w$ is not a duplication. Because $Del = \emptyset$ and $Xpos = \emptyset$, the only remaining possibility is to have an inversion. Let $z \in Inv$ be the string inverted in w' in order to obtain w, i.e. $w' = v_1 z v_2$ and $w = v_1 mi(z) v_2$ for some $v_1, v_2 \in V^*$. If $z \in V$, then $w' = w$ in contrast to our choice. If $mi(z)$ contains ab, then w' contains the subword ba in contrast to $w' \in L$. Therefore $mi(z) = \alpha a$ or $mi(z) = b\alpha$ or $mi(z) = b\alpha a$ hold for some $\alpha \in \{c, d, e\}$. In the former two cases w' contains ba, in the latter case w' contains aa and bb. Therefore in any case we get a contradiction to $w' \in L$.

Consequently, no operation can be used in the last step of producing a string w as above, and this completes the proof. □

A language L is called *strictly bounded* if and only if there are pairwise different letters $a_1, a_2, \ldots, a_n$ such that $L \subseteq a_1^* a_2^* \ldots a_n^*$.

Theorem 3.2.2 *A strictly bounded language can be generated by a non-deleting evolutionary grammar if and only if it is regular.*

Proof. In [50] it is shown that a strictly bounded language $L \subseteq a_1^* a_2^* \ldots a_n^*$ is regular if and only if there is an integer $r \in \mathrm{N}$, finite sets $F_{i,j} \subseteq \mathrm{N}$ and integers $m_{i,j} \in \mathrm{N}$, $1 \leq i \leq n$, $1 \leq j \leq r$, such that

$$\begin{aligned} L \;=\; & \bigcup_{j=1}^{r} \{a_1^{r_{1,j}+s_1 m_{1,j}} a_2^{r_{2,j}+s_2 m_{2,j}} \ldots a_n^{r_{n,j}+s_n m_{n,j}} \mid \\ & r_{i,j} \in F_{i,j}, s_i \in \mathrm{N} \text{ for } 1 \leq i \leq n\}. \end{aligned}$$

For $1 \leq i \leq n$, let m_i be the smallest common multiple of the integers $m_{i,1}$, $m_{i,2}$, …, $m_{i,r}$ different from 0 (or $m_i = 0$ if $m_{i,j} = 0$ for all j),

$$t = \max\{s : s \in F_{i,j}, 1 \leq i \leq n, 1 \leq j \leq r\}.$$

Now we consider the evolutionary grammar

$$EG = (\{a_1, a_2, \ldots a_n\}, A, \emptyset, \emptyset, \emptyset, \{a_i^{t \cdot m_i} \mid 1 \leq i \leq n\})$$

with

$$A = \{a_1^{u_1} a_2^{u_2} \ldots a_n^{u_n} \mid u_i \leq 2tm_i \text{ for } 1 \leq i \leq n\} \cap L.$$

Let

$$w = a_1^{r_{1,j}+s_1m_{1,j}} a_2^{r_{2,j}+s_2m_{2,j}} \dots a_n^{r_{n,j}+s_nm_{n,j}} \tag{3.1}$$

be a word in L. If a duplication of $a_i^{tm_i}$ for some i, $1 \le i \le n$, can be applied to w then $r_{i,j} + s_im_{i,j} \ge tm_i$. Hence, by $m_i = q_jm_{i,j}$ for some $q_j \in \mathbf{N}$, the word

$$\begin{aligned} w' \quad = \quad & a_1^{r_{1,j}+s_1m_{1,j}} a_2^{r_{2,j}+s_2m_{2,j}} \dots a_{i-1}^{r_{i-1,j}+s_{i-1}m_{i-1,j}} a_i^{r_{i,j}+(s_i+tq_i)m_{i,j}} \\ & a_{i+1}^{r_{i+1,j}+s_{i+1}m_{i+1,j}} \dots a_n^{r_{n,j}+s_nm_{n,j}} \end{aligned}$$

obtained by the duplication belongs to L, too. Since we start the generation of $L(EG)$ with words from $A \subseteq L$ and the application of duplications to words of L yields words of L, we get $L(EG) \subseteq L$.

We now prove the converse inclusion by induction on the length of words. Obviously, the shortest words in L belong to A and $A \subseteq L(EG)$ holds by definition. Thus let us consider a word of the form given in (3.1) which is not contained in A and assume that all words $w'' \in L$ which are shorter than w belong to $L(EG)$. Because $w \notin A$, there is a number i, $1 \le i \le n$ such that $r_{i,j} + s_im_{i,j} > 2tm_i$. By $r_{i,j} \le t \le tm_i$, we get $s_im_{i,j} > tm_i = tq_im_{i,j}$. Thus the word

$$\begin{aligned} w'' \quad = \quad & a_1^{r_{1,j}+s_1m_{1,j}} a_2^{r_{2,j}+s_2m_{2,j}} \dots a_{i-1}^{r_{i-1,j}+s_{i-1}m_{i-1,j}} a_i^{r_{i,j}+(s_i-tq_i)m_{i,j}} \\ & a_{i+1}^{r_{i+1,j}+s_{i+1}m_{i+1,j}} \dots a_n^{r_{n,j}+s_nm_{n,j}} \end{aligned}$$

is in L. By assumption $w'' \in L(EG)$. Because $w'' \Longrightarrow_{Dup} w$, we obtain $w \in L(EG)$.

Let $EG = (V, A, \emptyset, Inv, Xpos, Dup)$ be a non-deleting evolutionary grammar generating a strictly bounded language. Obviously, one may assume that $Inv = Xpos = \emptyset$ without modifying the language generated by EG. Now it is easy to see that the language generated by EG is regular. $\square$

We conjecture that Theorem 3.2.2 is valid for arbitrary evolutionary grammars as well. Next theorem proves this assertion for the unary alphabet.

Theorem 3.2.3 *A language over the unary alphabet is regular iff it is generated by an evolutionary grammar.*

Proof. By the first part of the previous proof it suffices to prove that each language over the unary alphabet generated by an evolutionary grammar is regular.

Consider an evolutionary grammar $EG = (\{a\}, Del, Inv, Xpos, Dup, A)$. Without loss of generality we may assume that $Inv = Xpos = \emptyset$. Let

$$\begin{aligned} \#(Del) &= \{|x| \mid x \in Del\} = \{d_1, d_2, \dots d_m\} \\ \#(Dup) &= \{c_1, c_2, \dots c_n\} \end{aligned}$$

and

$$p = gcd(d_1, d_2, \dots, d_m, c_1, c_2, \dots c_n)$$

Here *gcd* means the greatest common divisor. It is known that

$$p = \sum_{i=1}^{n} k_i c_i + \sum_{i=1}^{m} q_i d_i,$$

for some integers k_i, q_j, $1 \leq i \leq n$, $1 \leq j \leq m$. Moreover, one can choose $k_i \geq 0$ and $q_j \leq 0$, for all $1 \leq i \leq n$, $1 \leq j \leq m$.

If $L(EG)$ is finite, then it is obviously regular. If $L(EG)$ is an infinite set, there are $t_i, 1 \leq i \leq s$, $s \leq p$, such that

$$L(EG) = F \cup \bigcup_{i=1}^{s} \{a^{t_i + kp} | k \geq 0\},$$

for some finite set F. Consequently, $L(EG)$ is regular which completes the proof. □

A statement analogous to Theorem 3.2.2 does not hold for context-free languages because the strictly bounded context-free language $\{a^n b^n \mid n \geq 1\}$ cannot be generated by an evolutionary grammar. Indeed, let us assume the contrary and let $EG = (\{a, b\}, A, Del, Inv, Xpos, Dup)$ be an evolutionary grammar generating L. Obviously, $Dup \neq \emptyset$. For each $w \in Dup$ we distinguish three cases:

1. $w \in a^+$. Then, for large enough n, $a^n b^n \Longrightarrow_{Dup} a^{n+|w|} b^n$, contradiction.

2. Analogously for $w \in b^+$.

3. $w \in a^*abb^*$. Then a^nb^n may provide, by duplication, a word having two factors ab, contradiction.
Consequently, L cannot be generated by any evolutionary grammar.

We do not know whether or not Theorem 3.2.2 also holds for bounded languages, where a language L is *bounded* iff there are the strings $w_1, w_2, \ldots, w_n$ such that $L \subseteq w_1^*w_2^* \ldots w_n^*$.

In the view of the previous theorems it is of interest to look for features to be added to an evolutionary grammar in order to generate all regular languages. A squeezing mechanism in the form of a terminal alphabet is too powerful, since we get all recursively enumerable languages in this way as stated by the next theorem.

Theorem 3.2.4 *Each recursively enumerable language can be expressed as the intersection of a language in $\mathcal{L}(EG)$ with a regular language.*

Proof. Let $G = (N, T, S, P)$ be a phrase-structure grammar. We may assume (see [48]) that $N = \{S, A, B, C\}$ and

$$P = \{S \longrightarrow x_i | 1 \leq i \leq n, \text{ for some } n \geq 1\} \cup \{ABC \longrightarrow \varepsilon\}$$

We consider the evolutionary grammar

$$EG = (V, A, Del, Inv, Xpos, Dup)$$

with

$$\begin{aligned}
V &= N \cup T \cup \{\lhd, \rhd, \dashv\} \cup \{\perp_i | 1 \leq i \leq n\}, \\
Del &= \{ABC, \dashv\} \cup \{\perp_i S \lhd \ \rhd | 1 \leq i \leq n\} \cup \{\lhd x_i \perp_i \rhd \dashv | \\
&\quad\ 1 \leq i \leq n\}, \\
Inv &= \{\lhd x_i \perp_i | 1 \leq i \leq n\} \cup \{S \perp_i mi(x_i) | 1 \leq i \leq n\}, \\
Xpos &= \{\lhd x_i \perp_i \ \rhd | 1 \leq i \leq n\}, \\
Dup &= \{\lhd x_i \perp_i \rhd \dashv | 1 \leq i \leq n\}, \\
A &= \{\lhd x_1 \perp_1 \rhd \dashv \lhd x_2 \perp_2 \rhd \dashv \ldots \lhd x_n \perp_n \rhd \dashv S\}
\end{aligned}$$

Firstly, we shall prove that $L(G) \subseteq L(EG) \cap T^*$. More precisely, we shall prove that

$S \Longrightarrow_G^* y$ implies $\lhd x_1 \perp_1 \rhd \dashv \lhd x_2 \perp_2 \rhd \dashv \ldots \lhd x_n \perp_n \rhd \dashv y \in L(EG)$

Let $S \Longrightarrow_G^m y$ be a derivation in $m \geq 0$ steps. If $m = 0$, the assertion is trivially true. We assume the assertion true for any $k < m$ and prove it for m. Consider $S \Longrightarrow_G^{m-1} z \Longrightarrow y$. By the induction hypothesis, $\lhd x_1 \perp_1 \rhd \dashv \lhd x_2 \perp_2 \rhd \dashv \ldots \lhd x_n \perp_n \rhd \dashv z \in L(EG)$.

If the rule used at the last step was $ABC \longrightarrow \varepsilon$, then

$$\lhd x_1 \perp_1 \rhd \dashv \lhd x_2 \perp_2 \rhd \dashv \ldots \lhd x_n \perp_n \rhd \dashv z \Longrightarrow_{Del}$$
$$\lhd x_1 \perp_1 \rhd \dashv \lhd x_2 \perp_2 \rhd \dashv \ldots \lhd x_n \perp_n \rhd \dashv y$$

If the rule used at the last step was $S \longrightarrow x_i$, for some i, that is $z = z_1 S z_2, y = z_1 x_i z_2$, we consider the following derivation in EG:

$$\lhd x_1 \perp_1 \rhd \dashv \lhd x_2 \perp_2 \rhd \dashv \ldots \lhd x_n \perp_n \rhd \dashv$$
$$z \Longrightarrow_{Dup} \lhd x_1 \perp_1 \rhd \dashv \lhd x_2 \perp_2 \rhd \dashv \ldots \lhd x_i \perp_i \rhd \dashv \lhd x_i \perp_i \rhd \dashv \ldots$$
$$\lhd x_n \perp_n \rhd \dashv z \Longrightarrow_{Del} \lhd x_1 \perp_1 \rhd \dashv \lhd x_2 \perp_2 \rhd \dashv \ldots$$
$$\lhd x_i \perp_i \rhd \dashv \lhd x_i \perp_i \rhd \ldots \lhd x_n \perp_n \rhd \dashv z \Longrightarrow_{Xpos} \lhd x_1 \perp_1 \rhd$$
$$\dashv \lhd x_2 \perp_2 \rhd \dashv \ldots \lhd x_i \perp_i \rhd \dashv \ldots \lhd x_n \perp_n \rhd \dashv z_1 S \lhd x_i \perp_i \rhd z_2$$
$$\Longrightarrow_{Inv} \lhd x_1 \perp_1 \rhd \dashv \lhd x_2 \perp_2 \rhd \dashv \ldots \lhd x_i \perp_i \rhd \dashv \ldots \lhd x_n \perp_n \rhd \dashv$$
$$z_1 S \perp_i mi(x_i) \lhd \ \rhd z_2 \Longrightarrow_{Inv} \lhd x_1 \perp_1 \rhd \dashv \lhd x_2 \perp_2 \rhd \dashv \ldots$$
$$\lhd x_i \perp_i \rhd \dashv \ldots \lhd x_n \perp_n \rhd \dashv z_1 x_i \perp_i S \lhd \ \rhd z_2 \Longrightarrow_{Del}$$
$$\lhd x_1 \perp_1 \rhd \dashv \lhd x_2 \perp_2 \rhd \dashv \ldots \lhd x_i \perp_i \rhd \dashv \ldots \lhd x_n \perp_n \rhd \dashv z_1 x_i z_2$$

Now for any $y \in L(G)$ there exists the derivation in EG

$$\lhd x_1 \perp_1 \rhd \dashv \lhd x_2 \perp_2 \rhd \dashv \ldots \lhd x_n \perp_n \rhd \dashv S \Longrightarrow_{EG}^*$$
$$\lhd x_1 \perp_1 \rhd \dashv \lhd x_2 \perp_2 \rhd \dashv \ldots \lhd x_n \perp_n \rhd \dashv y \Longrightarrow_{Del}^* y$$

We shall discuss some considerations which lead to the conclusion $L(EG) \cap T^* \subseteq L(EG)$. For sake of simplicity, denote by $\alpha = \lhd x_1 \perp_1 \rhd \dashv \lhd x_2 \perp_2 \rhd \dashv \ldots \lhd x_n \perp_n \rhd \dashv$. It is easy to see that whenever $\alpha S \Longrightarrow_{EG}^* \beta$, with $\beta \in (N \cup T)^*$ we have also $\alpha S \Longrightarrow_{EG}^* \alpha\beta \Longrightarrow_{Del}^* \beta$.

Therefore, it suffices to prove that

$$\alpha\beta \Longrightarrow_{EG}^* \alpha\gamma \text{ implies } \beta \Longrightarrow_G^* \gamma,$$

where $\beta, \gamma \in (N \cup T)^*$. This results from the following remarks:

- Each deletion of a substring ABC corresponds to an application of the rule $ABC \longrightarrow \varepsilon$.

- Let $\beta = \beta_1\beta_2$ and $\alpha\beta \Longrightarrow_{Xpos} \beta_1 \lhd x_i\bot_i \rhd \beta_2$ be the result of transposing the segment $\lhd x_i\bot_i \rhd \beta_2$. The symbols $\lhd, \bot_i, \rhd$, not in $N \cup T$ can be removed only if one of the following cases holds:

    1. Roughly speaking, the erasing of the symbols $\lhd, \bot_i, \rhd$, requires that the segment $\lhd x_i\bot_i \rhd$ to be preceded by the symbol S. The overall effect is the substitution of S by x_i. Formally, $\alpha\beta_1 \Longrightarrow^*_{EG} \alpha\beta'_1 S$ and $\alpha\beta'_1 S \lhd x_i\bot_i \rhd \beta_2 \Longrightarrow^*_{EG} \alpha\beta'_1 x_i\beta_2$. This case can be covered in G by applying the rule $S \longrightarrow x_i$.
    2. The symbols $\lhd, \bot_i, \rhd$ can be cancelled if $\alpha mi(x_i) \Longrightarrow^*_{EG} \alpha S$ as well. But in this case, the transposition has no further effect.

□

3.2.2 Decidability Properties

The last theorem has a series of consequences regarding some decision problems of evolutionary grammars. In the view of biological interpretation of evolutionary grammars given in the beginning of this section, some of these decision problems might also have some biological relevance. Thus, the wellknown membership problem asks whether or not a given genome might appear from an initial set of genomes by evolution. Also the following matters appear to be of interest from the computational biology point of view:

1. Is the world generated by a given evolutionary grammar finite or infinite?

2. Are there common genomes in two given worlds?

3. Does the world generated by an evolutionary grammar contain all genomes which would support life? Or all genomes, no matter they would support life?

These problems, formally stated in the framework of our grammatical formalism, shall be discussed in the following.

Theorem 3.2.5 *The following problems are undecidable for the class of evolutionary grammars:*

1. The membership problem.

2. The inclusion problem.

3. The intersection problem. (Is the intersection of the languages generated by two given evolutionary grammars empty?)

4. Is R a subset of L for R being a regular language and L a language generated by an evolutionary grammar ?

Proof. Let $G_i = (N_i, T_i, S_i, P_i)$, $i = 1,2$, be two phrase-structure grammars with $N_1 \cap N_2 = \emptyset$, and $EG_i = (V_i, A_i, Del_i, Inv_i, Xpos_i, Dup_i)$, $i = 1,2$, the evolutionary grammars constructed as in the previous proof such that $V_1 \cap V_2 = T_1 \cap T_2$.

1. It is obvious that for each $x \in T_1^*$,

$$x \in L(EG_1) \text{ iff } x \in L(G_1),$$

hence the membership problem is undecidable.

2. Clearly, $L = \{x\}$ is a language that can be generated by an evolutionary grammar. Since

$$L \subseteq L(EG_1) \text{ iff } x \in L(G_1)$$

the undecidability status of the inclusion problem follows.

3. Observe that

$$L(EG_1) \cap L(EG_2) \neq \emptyset \text{ iff } L(G_1) \cap L(G_2) \neq \emptyset,$$

therefore the intersection problem is undecidable as well.

4. Take $R = T_1^*$ and $L = L(EG_1)$. The equivalence

$$R \subseteq L \text{ iff } L(G_1) = T_1^*$$

implies the last assertion of the theorem. □

Theorem 3.2.6 *It is not decidable whether or not an arbitrary given context-free language can be generated by an evolutionary grammar.*

Proof. The proof is an usual reduction to the Post's Correspondence Problem. Take two arbitrary n-tuples of nonempty strings over the alphabet $\{a, b\}$,

$$x = (x_1, x_2, \ldots, x_n),$$

$$y = (y_1, y_2, \ldots, y_n).$$

Then, consider the languages

$$L_z = \{ba^{t_1}ba^{t_2}\ldots ba^{t_k}cz_{t_k}\ldots z_{t_2}z_{t_1} | k \geq 1\}$$

for $z \in \{x, y\}$,

$$L_s = \{w_1cw_2cmi(w_2)cmi(w_1) | w_1, w_2 \in \{a, b\}^*\}$$

$$L(x, y) = \{a, b, c\}^* - (L_x\{c\}mi(L_y) \cap L_s)$$

It is known that $L(x, y)$ is a context-free language. For every solution $(i_1, i_2, \ldots, i_k)$ of $PCP(x, y)$ the strings

$$ba^{i_1}ba^{i_2}\ldots ba^{i_k}cx_{i_k}\ldots x_{i_2}x_{i_1}cmi(y_{i_1})mi(y_{i_2})\ldots mi(y_{i_k})$$
$$ca^{i_k}b\ldots ba^{i_2}ba^{i_1}b$$

are not in $L(x, y)$. On the other hand, $\{a, b\}^* \subseteq L(x, y)$.

Clearly, when $L(x, y) = \{a, b, c\}^*$, then $L(x, y)$ can be generated by the evolutionary grammar:

$$EG = (\{a, b, c\}, \{abc\}, \{a, b, c\}, \emptyset, \{a, b, c\}, \{a, b, c\})$$

Now, it is sufficient to prove that when $L(x, y) \neq \{a, b, c\}^*$, then $L(x, y)$ is cannot be generated by any evolutionary grammar and we will do that in the sequel.

Let us suppose that $L(x, y) = L(EG)$, for some $EG = (\{a, b, c\}, A, Del, Inv, Xpos, Dup)$. We choose a solution $(i_1, i_2, \ldots i_k)$ such that $|x_{i_k}x_{i_{k-1}}\ldots x_{i_1}| > max\{|w| \mid w \in A\}$. For $\{a, b\}^* \subseteq L(EG)$, there exists a word $w \in A$ such that

$$w \Longrightarrow^*_{EG} mi(y_{i_1})mi(y_{i_2})\ldots mi(y_{i_k}) \in L(EG)$$

By the choice of the solution $(i_1, i_2, \ldots, i_k)$ the word

$$z = ba^{i_1}ba^{i_2}\ldots ba^{i_k}cx_{i_k}\ldots x_{i_2}x_{i_1}cwca^{i_k}b\ldots ba^{i_2}ba^{i_1}b$$

is in $L(EG)$.

Therefore,

$$z \Longrightarrow^*_{EG} ba^{i_1}ba^{i_2}\ldots ba^{i_k}cx_{i_k}\ldots x_{i_2}x_{i_1}cmi(y_{i_1})mi(y_{i_2})\ldots mi(y_{i_k})$$
$$ca^{i_k}b\ldots ba^{i_2}ba^{i_1}b,$$

contradiction, and the proof is complete. □

Theorem 3.2.7 *1. It is decidable whether or not the language generated by a given evolutionary grammar is finite.*

2. The problem "$L(EG) = V^*$?" is decidable for a given non-deleting evolutionary grammar EG.

3. The membership problem is decidable for non-deleting evolutionary grammars.

Proof. The first assertion is immediately true. Indeed, for a given evolutionary grammar $EG = (V, A, Del, Inv, Xpos, Dup)$, $L(EG)$ is infinite if, and only if, $Sub(L(EG_1)) \cap Dup \neq \emptyset$, where $EG_1 = (V, A, Del, Inv, Xpos, \emptyset)$ and $Sub(X)$ means the set of all factors of the strings in X. Because the set $L(EG_1)$ is finite the proof of the first assertion is complete.

The second and the third items follow from [96], since any evolutionary grammar is actually a pure grammar. □

By Theorem 3.2.2 the following open decision problems are still of interest:

- Does a given evolutionary grammar generate a regular language?
- Can a given regular language be generated by an evolutionary grammar?

3.2.3 Some Closure Properties

Theorem 3.2.8 *The family $\mathcal{L}(EG)$ is not closed under union, concatenation, morphisms, intersection with regular sets and intersection.*

Proof. We consider the languages:

$$L_1 = \{x \in \{a,b\}^+ \mid \mid x \mid_a = \mid x \mid_b\}$$

$$L_2 = a^+b^+$$

Both of them can be generated by evolutionary grammars. Moreover, L_2 is regular.

The same reasoning may be used in order to prove the non-closure under union and concatenation. We shall discuss it in the case of concatenation. We claim that L_1L_2 is not in $\mathcal{L}(EG)$. Indeed, let $EG = (\{a,b\}, A, Del, Inv, Xpos, Dup)$ be an evolutionary grammar generating L_1L_2. Recall that the Parikh vector associated to a word x in $\{a,b\}^*$, denoted by $\Psi(x)$, is $\Psi(x) = (\mid x \mid_a, \mid x \mid_b)$. Note that the transpositions and inversions do not change the Parikh vector associated to a word.

Since there are words in $L(EG)$ whose Parikh vector (n_1, n_2) satisfies the requirement that $n_1 - n_2$ is arbitrarily large, at least one of Del or Dup has to contain words x with $\mid x \mid_a \neq \mid x \mid_b$. We shall analyse the case when Dup contains such words. A similar analysis for the case when Del contains such words is left to the reader. Let $x \in Dup$ be such a word.

If $\mid x \mid_a < \mid x \mid_b$, then the following derivation is possible in EG

$$xa^{|x|_b - |x|_a}ab \Longrightarrow_{EG} xxa^{|x|_b - |x|_a}ab$$

which implies $xxa^{|x|_b-|x|_a}ab \in L(EG)$, contradiction.

If $\mid x \mid_a > \mid x \mid_b$, then the following derivation is possible in EG

$$xb^{|x|_a - |x|_b}ab \Longrightarrow_{EG} xxb^{|x|_a - |x|_b}ab$$

which implies $xxb^{|x|_a-|x|_b}ab \in L(EG)$, contradiction. Therefore, L_1L_2 cannot be generated by EG.

Define the morphism $h : \{a,b,c,d\}^* \longrightarrow \{a,b\}^*$ by $h(a) = h(c) = a$, $h(b) = h(d) = b$. Since $L_1 \cup c^+d^+ \in \mathcal{L}(EG)$ and $h(L_1 \cup c^+d^+) = L_1 \cup L_2$, it follows that $\mathcal{L}(EG)$ is not closed under morphisms.

The non-closure under intersection with regular sets and intersection can be settled in the same way. The language $L_1 \cap L_2 = \{a^nb^n \mid n \geq 1\}$ cannot be generated by any evolutionary grammar by Theorem 3.2.1. □

3.2.4 Evolutionary Grammars and the Structural Language of Nucleic Acids

Theorem 3.2.9 *1. There is an evolutionary grammar that generates L_{DNA}.*

2. It is decidable whether or not a given non-deleting evolutionary grammar generates L_{DNA}.

Proof. 1. We consider the following evolutionary grammar:

$$EG = (V_{DNA}, A, , Inv, Xpos, Dup)$$

with

$$\begin{aligned} Inv &= Xpos = Dup = \{AT, CG\} \\ A &= \{AT, CG, ATCG\} \end{aligned}$$

We claim that $L(EG) = L_{DNA}$. It is easy to verify that all strings of length at most 4 in L_{DNA} are also in $L(EG)$. Assume that all strings $x \in L_{DNA}, |x| \leq 2n$ are in $L(EG)$, for some $n \geq 2$, and consider a string $w \in L_{DNA}$ with $|w| = 2n + 2$. Clearly, $w = x_1 a\bar{a} x_2$, for some $a \in V_{DNA}$. By the choice of Inv it suffices to consider only the cases $a = A$ and $a = C$. Furthermore, since $CGAT \in L(EG)$, we consider the case $a = A$ only. Note that $x_1x_2 \in L_{DNA}$, $|x_1x_2| = 2n$. By our hypothesis $x_1x_2 \in L(EG)$.

If $AT \Longrightarrow_{EG}^{*} x_1x_2$, then the following derivation is also possible in EG:

$$AT \Longrightarrow_{Dup} ATAT \Longrightarrow_{EG}^{*} ATx_1x_2 \Longrightarrow_{Xpos} x_1ATx_2$$

If $CG \Longrightarrow_{EG}^{*} x_1x_2$, then the following derivation is also possible in EG:

$$ATCG \Longrightarrow_{EG}^{*} ATx_1x_2 \Longrightarrow_{Xpos} x_1ATx_2$$

If $ATCG \Longrightarrow_{EG}^{*} x_1x_2$, then the following derivation is also possible in EG:

$$ATCG \Longrightarrow_{Dup} ATATCG \Longrightarrow_{EG}^{*} ATx_1x_2 \Longrightarrow_{Xpos} x_1ATx_2$$

Consequently, $x \in L(EG)$. We finish the proof of the first item by observing that $L(EG) \subseteq L_{DNA}$.

2. Let $EG = (V_{DNA}, A, \emptyset, Inv, Xpos, Dup)$ be a non-deleting evolutionary grammar. We claim that $L_{DNA} \subseteq L(EG)$ if, and only if, all strings in L_{DNA} of length at most $2k$ can be generated by EG. Here $k = max\{|x| \mid x \in A\}$. It remains to prove that the aforementioned condition is sufficient. We shall prove that each $x \in L_{DNA}$, $|x| = 2k + 2$ is in $L(EG)$. There are two possibilities:

(i) $x = x_1x_2$, $x_1, x_2 \in L_{DNA}$.
Since $|x| = 2k + 2$, one of x_1, x_2 is of length bigger than k. Assume that $|x_1| > k$. Then, there is $y \in A$ such that

$$y \Longrightarrow^+_{EG} x_1$$

The word $yx_2 \in L(EG)$ because $|yx_2| \leq 2k$. But

$$yx_2 \Longrightarrow_{EG} x_1x_2,$$

hence $x \in L(EG)$.

(ii) $x = ax_3\bar{a}$, for some $a \in V_{DNA}$.
There is $z \in A$ such that $z \Longrightarrow^+_{EG} x_3$. The string $az\bar{a} \in L(EG)$ and $az\bar{a} \Longrightarrow^+_{EG} x$, hence $x \in L(EG)$. Inductively, all strings in L_{DNA} can be generated by EG.

We claim that if $L_{DNA} \subseteq L(EG)$ we have $L(EG) \subseteq L_{DNA}$ if, and only if, $A, Xpos$, $Dup \subseteq L_{DNA}$ and Inv contains only words that can be reduced to palindromes. All these conditions can be algorithmically checked. The "if" part is immediate. In order to prove the "only if" part, we shall consider each condition separately. Obviously, if A contains words not in L_{DNA} the same is true for $L(EG)$.

Assume that $x \in Dup$ and let xy be in $L_{DNA} \subseteq L(EG)$. Because xxy has to be in L_{DNA} it follows that $x \in L_{DNA}$.

Let $x \in Xpos \setminus L_{DNA}$ and y be the shortest word such that $xy \in L_{DNA}$. Note that such a word always exists. Let z be the reduced word associated to x. Assume that z starts with a and y ends with b. Consider $xyc\bar{c} \in L_{DNA}$ such that $c \neq \bar{a}, c \neq \bar{b}$. We get

$$xyc\bar{c} \Longrightarrow_{Xpos} ycx\bar{c}$$

and

$$ycx\bar{c} \in L_{DNA} \text{ iff } ycz\bar{c} \in L_{DNA}$$

The only possible reduction that can take place within $ycz\bar{c}$, concerns the string $z\bar{c}$, hence $ycz\bar{c} \notin L_{DNA}$.

Now, let $x \in Inv$ and z be its reduced word. Again y is the shortest word such that $xy \in L_{DNA}$. We have $zy \in L_{DNA}$ and $mi(x)y \in L_{DNA}$. Because the reduced word associated to $mi(x)$ is $mi(z)$, it follows that $mi(z) = z$ if $mi(z)y \in L_{DNA}$. Now the proof is complete. □

3.2.5 Descriptional Complexity

In this section we consider the descriptional (syntactic) complexity of languages generated by evolutionary grammars following [40]. We are interested in the minimal number of axioms and operations, respectively, and the maximal length of the words associated with an operation. Formally, for an evolutionary grammar $G = (V, A, Del, Inv, Xpos, Dup)$, we set

$$\begin{aligned} a(G) &= card(A), \\ o(G) &= card(Del) + card(Inv) + card(Xpos) + card(Dup), \\ l(G) &= \max\{|w| \mid w \in Del \cup Inv \cup Xpos \cup Dup\} \end{aligned}$$

and extend these measures to a language L generated by an evolutionary grammar by

$$\begin{aligned} a(L) &= \min\{a(G) \mid L = L(G), G \text{ is an evolutionary grammar}\}, \\ o(L) &= \min\{o(G) \mid L = L(G), G \text{ is an evolutionary grammar}\}, \\ l(L) &= \min\{l(G) \mid L = L(G), G \text{ is an evolutionary grammar}\}. \end{aligned}$$

Theorem 3.2.10 *A language L is finite if and only if $o(L) = 0$.*

Proof. Let L be a finite language, and let V be the set of symbols occurring in at least one word of L. Then $L = L(G)$ for the evolutionary grammar $G = (V, L, \emptyset, \emptyset, \emptyset, \emptyset)$. Since $o(G) = 0$ we obtain $o(L) = 0$.

If $o(L) = 0$ for some language L, then there is an evolutionary grammar $G = (V, A, \emptyset, \emptyset, \emptyset, \emptyset)$ with $L = L(G)$. By $L(G) = A$, G generates a finite language which proves that L is finite. □

The measure $o(G)$ corresponds to the number of productions in a (usual) Chomsky grammar. The context-free languages form an infinite hierarchy with respect to the number of productions (see [54]). Furthermore, the measure $l(G)$ corresponds to the radius of an H system which is grammatical device based on splicing. With respect to the radius the languages generated by H systems form an infinite hierarchy, too (see [104]). In this section we shall prove analogous assertions for the measures for evolutionary grammars introduced above.

Theorem 3.2.11 *For any measure $d \in \{a, o, l\}$ and any natural number $r \geq 1$, there is a language L generated by an evolutionary grammar such that $d(L) = r$.*

Proof. We consider the language

$$L = \bigcup_{i=1}^{n} L_i \quad \text{where} \quad L_i = \{(ba^ib)^m \mid m \geq 0\} \text{ for } 1 \leq i \leq n.$$

Because L is generated by the evolutionary grammar

$$G = (\{a, b\}, \{ba^ib \mid 1 \leq i \leq n\}, \emptyset, \emptyset, \emptyset, \{ba^ib \mid 1 \leq i \leq n\})$$

with $a(G) = n$, $o(G) = n$ and $l(G) = n + 2$, we obtain

$$a(L) \leq n, \quad o(L) \leq n \quad \text{and} \quad l(L) \leq n + 2. \tag{3.2}$$

Now let us assume that $H = (V, A, Del, Inv, Xpos, Dup)$ is an evolutionary grammar with $L(H) = L$. If there is a derivation $w' \Longrightarrow w$ in H with $w' \in L_i$, $w \in L_j$, $1 \leq i, j \leq n$ and $i \neq j$, then there also is a derivation $w'ba^ib \Longrightarrow wba^ib$. Since $w'ba^ib \in L_i \subset L$ and $wba^ib \notin L$, we get a contradiction. Thus, for any i, $1 \leq i \leq n$, $A \cap L_i$ has to be a non-empty set. Therefore $a(H) \geq n$ for any evolutionary grammar H with $L(H) = L$ which implies $a(L) \geq n$. By (3.2), we obtain $a(L) = n$.

Let $a = \max\{|z| \mid z \in A\}$. We consider a word $w \in L_i$, $1 \leq i \leq n$, with $|w| \geq a + 1$. Let $w = (ba^ib)^l$. By the length of w there is a word $w' \in L$ with $w' \Longrightarrow w$ and $w' \neq w$. By the above considerations $w' \in L_i$, too, say $w' = (ba^ib)^k$ for some k.

If $w' \Longrightarrow_{Inv} w$ or $w' \Longrightarrow_{Xpos} w$, then $|w'| = |w|$ and hence $w' = w$ in contrast to the choice of w'.

Let us assume that $w' \Longrightarrow_{Dup} w$. Then $w' = w_1xw_2$ and $w = w_1xxw_2$ for some $w_1, w_2 \in V^*$, $x \in V^+$. Thus

$$|\, x\,|_a = |\, w\,|_a - |\, w'\,|_a = i(l-k) \quad \text{and } |\, x\,|_b = 2(l-k)\,.$$

Therefore

$$x \in \{a^r b(ba^i b)^{l-k-1} ba^s \mid r, s \geq 0, r+s = i\} \cup \{(ba^i b)^{l-k}\}\,.$$

If $x = a^r bba^s$, $r, s \geq 0$, $r+s = i$, i.e. $l = k+1$, then we can apply $x \in Dup$ to $ba^n bba^n b$ which yields $ba^{2n-i}b$ and hence $n = i$. If, in addition, $r > 0$ and $s > 0$, we can apply $x \in Dup$ to $ba^{n-1}bba^{n-1}bba^{n-1}b$ and obtain $ba^{2n-2-i}bba^{n-1}b$ from which $i = n-1$ follows. This contradicts $i = n$. Thus $x = a^r bba^s$ implies $r+s = n$ and $r = 0$ or $s = 0$.

We now define

$$\begin{aligned} M_j &= \{a^r b(ba^j b)^t ba^s \mid r, s \geq 0, r+s = j, t \geq 0\} \cup \\ &\quad \{(ba^j b)^t \mid t \geq 1\} \text{ for } 1 \leq j < n, \\ M_n &= \{a^r b(ba^n b)^t ba^s \mid r, s \geq 0, r+s = n, t \geq 0\} \cup \\ &\quad \{a^n bb, bba^n\} \cup \{(ba^n b)^t \mid t \geq 1\}. \end{aligned}$$

By the considerations above, we get $x \in M_i$.

Let $w' \Longrightarrow_{Del} w$. Then $w' = w_1xw_2$ and $w = w_1w_2$. By analogous arguments we can show that $x \in M_i$, again.

Thus $(Del \cup Dup) \cap M_i \neq \emptyset$ for $1 \leq i \leq n$. Furthermore, $M_i \cap M_j = \emptyset$ for $1 \leq i, j \leq n$ and $i \neq j$. Therefore $Del \cup Dup$ contains at least n elements and $o(H) \geq n$ holds for any evolutionary grammar H with $L(H) = L$. Hence $o(L) \geq n$. By (3.2), $o(L) = n$.

Moreover, for $1 \leq i \leq n$, $|x| \geq 2+i$ holds for any $x \in (Del \cup Dup) \cap M_i$. Thus $l(H) \geq n+2$ for any evolutionary grammar H with $L(H) = L$. Therefore $l(L) \geq n+2$ and, by (3.2), $l(L) = n+2$.

Hence the statement holds for $d \in \{a, o\}$, $r \geq 1$ and $d = l$, $r \geq 3$. It is easy to see that $l(\{a\}^+) = 1$ and $l(\{a^2\}^+) = 2$, and therefore the statement holds in the remaining cases, too. □

3.2.6 The Differentiation Function

The notion of a differentiation function of a grammar was firstly introduced in [29] for deterministic tabled Lindenmayer systems. It presents a measure for the number of objects which can be derived in a given grammar by a given number of derivation steps. Formally we obtain the following notion for evolutionary grammars [40].

Let $G = (V, A, Del, Inv, Xpos, Dup)$ be an evolutionary grammar. Then we define its *differentiation function*

$$f_G : \mathrm{N} \to \mathrm{N} \quad \text{by} \quad f_G(k) = card(L_k(G)),$$

where $L_k(G)$ consists of all strings obtained from A after exactly k mutations.

Example 3.2.1 *We consider the evolutionary grammars*

$$\begin{aligned} G_1 &= (\{a,b\}, \{aa\}, \emptyset, \emptyset, \emptyset, \{aa\})\,, \\ G_2 &= (\{a,b\}, \{aa\}, \emptyset, \{aa\}, \emptyset, \{aa\})\,, \\ G_3 &= (\{a,b\}, \{aab\}, \emptyset, \{aa\}, \{b\}, \{aa\})\,. \end{aligned}$$

Then, for $k \geq 1$,

$$\begin{aligned} L_k(G_1) &= \{a^{2k+2}\}\,, \\ L_k(G_2) &= \{a^2, a^4, \ldots, a^{2k+2}\}\,, \\ L_k(G_3) &= \{a^r b a^s \mid r + s = 2i, 1 \leq i \leq k\} \cup \{a^{2k+2}b\}\,, \end{aligned}$$

and thus

$$\begin{aligned} f_{G_1}(k) &= 1, \text{ for } k \geq 1\,, \\ f_{G_2}(k) &= k+1, \text{ for } k \geq 1\,, \\ f_{G_3}(k) &= 3 + 5 + \ldots (2k+1) + 1 = (k+1)^2, \text{ for } k \geq 1\,. \end{aligned}$$

We only show the statement concerning $L_k(G_3)$, the modifications for the other cases are obvious.

From the axiom aab of G_3 we can generate by inversion of aa the same word aab, by transposition of b the words baa, aba and by duplication of aa the word $aaaab$. Thus the statement holds for $k = 1$.

Now let $w \in L_k(G_3)$. By induction hypothesis, $w = a^r b a^s$ with $r+s = 2i$ for some i, $1 \leq i \leq k$ or $w = a^{2k+2}b$.

We first consider the former case. Since the transposition and inversion does not change the number of occurrences of a and b, we obtain by these operation a word $a^{r'}ba^{s'}$ with $r'+s' = 2i$. If we applied the duplication of aa we get $a^{r+2}ba^s$ or $a^r ba^{s+2}$. Because $r+s+2 = 2(i+1)$, in all cases the generated words have the desired form.

In the former case we generate from w a word of the $a^r ba^s$ with $r+s = 2k+2 = 2(k+1)$ or $a^{2k+4}b = a^{2(k+1)+2}b$, and all words have the desired form, again.

Moreover, these considerations also show that all words of the desired form are contained in $L_{k+1}(G_3)$.

We now give an upper bound for differentiation functions of evolutionary grammars.

Theorem 3.2.12 *For any evolutionary grammar G, there are constants c_1 and c_2 such that $f_G(k) \leq c_1 \cdot c_2^k$ for $k \geq 1$.*

Proof. Let $G = (V, A, Del, Inv, Xpos, Dup)$ be an evolutionary grammar. We set

$$d = \max\{|u| \mid u \in Dup\}\,, \quad a = card(A)\,, \quad b = \max\{|v| \mid v \in A\}\,.$$

Then, for any $k \geq 0$ and any word in $z \in L_k(G)$, $|z| \leq b + k \cdot d$. Thus

$$\begin{aligned} f_G(k+1) &\leq card(\{w \mid |w| \leq b+kd\}) \\ &= \sum_{i=0}^{b+kd} (card(V))^i = \frac{1}{card(V)-1} \cdot ((card(V))^{b+kd+1} - 1) \\ &\leq \frac{(card(V))^{b+1}}{card(V)-1}((card(V))^d)^k\,. \end{aligned}$$

By setting $c_1 = \frac{(card(V))^{b+1}}{card(V)-1}$ and $c_2 = (card(V))^d)$ the assertion follows. □

The following shows that the exponential upper bound is obtained for some evolutionary grammars.

Theorem 3.2.13 *For any natural number c, there is an evolutionary grammar G such that $f_G(k) = c^k$ for $k \geq 0$.*

Proof. Let c be given. We consider the evolutionary grammar

$$G = (V, \{a^c ba^c\}, \emptyset, \emptyset, \emptyset, \{a^c ba^i \mid 1 \leq i \leq c\}).$$

By induction on k we prove that

$$L_k(G) = \{a^c ba^{i_1+c} ba^{i_2+c} b \dots a^{i_k+c} ba^c \mid i_1, i_2, \dots i_k \in \{1, 2 \dots c\}\}$$

which implies

$$f_G(k) = card(L_k(G)) = c^k$$

for $k \geq 0$.

By definition, $L_0(G) = \{a^c ba^c\}$ and therefore the statement holds for $k = 0$.

Now let $w \in L_{k+1}(G)$. Then $w' \Longrightarrow w$ for some $w' \in L_k(G)$. By induction hypothesis $w' = a^c ba^{i_1+c} ba^{i_2+c} b \dots a^{i_k+c} ba^c$. Let w be derived from w' by duplication of some $a^c ba^i$, $1 \leq i \leq c$, where the duplication involves the j-th occurrence of b. Then

$$\begin{aligned}
w' &= a^c ba^{i_1+c} ba^{i_2+c} b \dots a^{i_j+c} ba^{i_{j+1}+c} b \dots a^{i_k+c} ba^c \\
&= a^c ba^{i_1+c} ba^{i_2+c} b \dots a^{i_j} a^c ba^i a^{i_{j+1}+c-i} b \dots a^{i_k+c} ba^c \\
&\Longrightarrow a^c ba^{i_1+c} ba^{i_2+c} b \dots a^{i_j} a^c ba^i a^n ba^i a^{i_{j+1}+c-i} b \dots a^{i_k+c} ba^c \\
&= a^c ba^{i_1+c} ba^{i_2+c} b \dots a^{i_j+c} ba^{i+c} ba^{i_{j+1}+c} b \dots a^{i_k+c} ba^c
\end{aligned}$$

which proves that w has the desired form. If we apply in succession the duplications of $a^c ba^{i_1}, a^c ba^{i_2}, \dots, a^c ba^{i_{k+1}}$ such that in any step the last b is involved, then we get $a^c ba^{i_1+c} ba^{i_2+c} b \dots ba^{i_{k+1}+c} ba^n \in L_{k+1}(G)$. Hence $L_{k+1}(G)$ contains all words of the considered form.

Thus the induction statement is shown for $k + 1$. □

We have shown that, for any exponential function f with a positive integer as exponent, there is a context-free evolutionary grammar G whose differentiation function is asymptotically equal to f. We now want to prove such a statement for polynomials.

Theorem 3.2.14 *For any natural number n, there is an evolutionary grammar G such that $f_G(k) = \Theta(k^n)$.*

Proof. For $n \in \{0,1,2\}$ the statement follows from Example 3.2.1. Let us assume that there is already an evolutionary grammar $G' = (V, A, Del, Inv, Xpos, Dup)$ with $f_{G'}(k) = \Theta(k^n)$. Without loss of generality we may suppose that $V \cap \{a,b\} = \emptyset$ and construct the evolutionary grammar

$$G = (V \cup \{a,b\}, \{aa\} \cdot A, Del, Inv, Xpos, Dup \cup \{aa\}),$$

where the sets $Del, Inv, Xpos, Dup$ are taken from G'. By induction we show that

$$L_k(G) = \{a^2\} \cdot L_k(G') \cup \{a^4\} \cdot L_{k-1}(G') \cup \cdots \cup \{a^{2k+2}\} \cdot L_0(G') . \quad (3.3)$$

By the construction of G, the statement holds for $k = 0$.

Let $w \in L_{k+1}(G)$. Then there is a word $w \in L_k(G)$ with $w \Longrightarrow w'$. Furthermore there exists an integer i, $1 \leq i \leq k$, such that $w = a^{2i+2}v$ for some word $v \in L_{k-i}(G')$. Let us apply an element $x \in Del \cup Inv \cup Xpos \cup Dup$ to w in order to get w'. Then we have to apply x to v and get $w = a^{2i+2}v \Longrightarrow a^{2i+2}v' = w'$ where $v' \in L_{k-i+1}(G')$. Hence $w' \in \{a^{2i+2}\}L_{k-i+1}(G') = \{a^{2i+2}\}L_{(k+1)-i}(H)$. If we apply the duplication of aa to w, we get

$$w = a^{2i+2}v \Longrightarrow a^{2i+4}v = a^{2(i+1)+2}v = w' \in \{a^{2(i+1)+2}\}L_{k-i}(G') .$$

This proves

$$L_{k+1}(G) \subseteq \{a^2\} \cdot L_{k+1}(G') \cup \{a^4\} \cdot L_k(G') \cup \cdots \cup \{a^{2(k+1)+2}\} \cdot L_0(G') .$$

On the other hand let

$$u' \in \{a^2\} \cdot L_{k+1}(G') \cup \{a^4\} \cdot L_k(G') \cup \cdots \cup \{a^{2(k+1)+2}\} \cdot L_0(G') .$$

Then there exists an integer i such that $u' = a^{2i+2}z'$ where $z' \in L_{(k+1)-i}(G)$. If $i \geq 1$, a duplication of aa in $u = a^{2i}z'$ gives u'. Because $u = a^{2i}z' \in \{a^{2(i-1)+2}\}L_{k-(i-1)}(G') \subseteq L_k(G)$ and $u \Longrightarrow u'$ we have $u' \in L_{k+1}(G)$.

If $i = 0$, then $k+1-i = k+1 \geq 1$ and there is a word $z \in L_k(G')$ with $z \Longrightarrow z'$ in G'. Hence $a^2z \Longrightarrow a^2z'$ in G and $a^2z \in \{a^2\}L_k(G') \subseteq L_k(G)$. Thus $a^2z' = u' \in L_{k+1}(G)$. Therefore

$$\{a^2\} \cdot L_{k+1}(G') \cup \{a^4\} \cdot L_k(G') \cup \cdots \cup \{a^{2(k+1)+2}\} \cdot L_0(G') \subseteq L_{k+1}(G) .$$

Therefore (3.3) is shown.

By (3.3) and the disjointness of the sets involved in the union,

$$\begin{aligned} f_G(k) &= card(L_k(G')) = \sum_{i=0}^{k} card(\{a^{2i+2}\}L_{k-i}(G') = \sum_{i=0}^{k} f_{G'}(k-i) \\ &= \sum_{i=0}^{k} f_G(i) = \Theta(k^{n+1}) \,. \end{aligned}$$

□

Without proof (use disjoint alphabets and unions of the involved sets) we add the following lemma on closure properties of the set of differentiation functions.

Lemma 3.2.1 *Let f and g be two differentiation functions of evolutionary grammars. Then their sum $f+g$ is a differentiation function, too.* □

The presented results give some upper and lower bounds for and some examples of differentiation functions; the characterization of the family of differentiation functions of evolutionary grammars is left as an open problem.

Finally we present two classes of evolutionary grammars with differentiation functions bounded by a constant or linear function.

Lemma 3.2.2 *If $G = (V, A, Del, Inv, Xpos, \emptyset)$, then there is a constant c such that $f_G(k) \leq c$ for $k \geq 0$.*

Proof. Obviously, because the set of duplications is empty, $L(G)$ is finite. Let c be the cardinality of $L(G)$. Then $f_G(k) = card(L_k(G)) \leq card(L(G)) = c$. □

Example 3.2.1 shows that there are evolutionary grammars with a non-empty set of duplications which also have a differentiation function bounded by a constant.

Before we present the other class we give the definition and a property of a number-theoretic function. For a set $A = \{a_1, a_2, \ldots, a_n\}$ of natural numbers we define the function

$$g_A(k) = card(\{\sum_{j=1}^{k} a_{i_j} \mid a_{i_j} \in A \text{ for } 1 \leq j \leq k\}) \,.$$

This function expresses the number of all distinct sums of k elements from A.

Lemma 3.2.3 *Let $A = \{a_1, a_2, \ldots, a_s\}$ with $0 \leq a_1 < a_2 < a_3 < \ldots < a_s$ and $m = \max\{a_{i+1} - a_i \mid 1 \leq i \leq s-1\}$. Then*

$$\lfloor \frac{a_s - a_1}{m} \cdot k \rfloor + 1 \leq g_A(k) \leq (a_s - a_1) \cdot k + 1.$$

Moreover, both bounds are optimal.

Proof. Obviously $a_1 \cdot k$ and $a_s \cdot k$ are the minimal and maximal number which can be obtained by addition of k numbers of A. Hence any sum S of interest satisfies $a_1 k \leq S \leq a_s k$. This implies the upper bound.

We now prove that any interval $I_i = [a_1 k + im, a_1 k + (i+1)m)$, $0 \leq i \leq \lfloor \frac{(a_s - a_1)k}{m} \rfloor - 1$, contains at least one sum of k numbers of A. Obviously, this holds for $i = 0$ by $a_1 k \in I_0$. Now let $c = \sum_{j=1}^{k} a_{i_j}$ be the maximal number in I_i which can be represented by sum of k numbers of A. Since $i \leq \lfloor \frac{(a_s - a_1)k}{m} \rfloor - 1$ we obtain $c \leq a_1 k + (\lfloor \frac{(a_s - a_1)k}{m} \rfloor - 1)m < a_s k$. Thus there exists an r such that $a_{i_r} < a_s$. Let $a_{i_r} = a_l$. Then we consider the sum

$$c' = \Big(\sum_{j=1}^{r-1} a_{i_j}\Big) + a_{l+1} + \Big(\sum_{j=r+1}^{k} a_{i_j}\Big)$$

of k numbers of A. Because $c' = c + (a_{l+1} - a_l) \leq c + m$ and c is maximal in I_i, we obtain $c' \in I_{i+1}$. Since we have $\lfloor \frac{(a_s - a_1)k}{m} \rfloor$ intervals and the additional sum $a_s k$ (which belongs to no interval), the lower bound follows.

The optimality of both bounds follows by considering $A = \{1, 2, 3, \ldots, s\}$ or $A = \{m, 2m, 3m, \ldots, sm\}$ (for some m). □

Lemma 3.2.4 *For any evolutionary grammar $G = (\{a\}, \{a^n\}, \emptyset, Inv, Xpos, Dup)$ where Dup contains a non-empty word (i.e. the underlying alphabet of the grammar is unary, there is only one axiom, no deletion and at least one non-empty duplication), $f_G(k) = \Theta(k)$ holds.*

Proof. First we assume that $Inv \cup Xpos = \emptyset$. The application of $v \in Dup$ to a^m leads to $a^{m+|v|}$. Thus taking $A = \{|v| \mid v \in Dup\}$, we obtain $g_A(k) = f_G(k)$ for $k \geq 1$. Hence the statement follows from Lemma 3.2.3.

Now let $Inv \cup Xpos \neq \emptyset$ and $\varepsilon \notin Dup$. We set $r = \min\{|z| \mid z \in Inv \cup Xpos\}$. If we apply an inversion or a transposition to a word w, then w is not changed by this operation because the underlying alphabet is unary. To any word $w \in L_k(G)$ with $|w| \geq r$ we can apply an inversion or a transposition, and thus w is also contained in $L_l(G)$ for $l \geq k$. Moreover, if Dup does not contain the empty word, all words from $L_r(G)$ have a length $\geq r$. It is easy to see that

$$g_{A\cup\{0\}}(k) \leq f_G(k+r) \leq card(L_r(G))g_{A\cup\{0\}}(k) \text{ for } k \geq 1\,.$$

Now the statement follows from Lemma 3.2.3.

The proof for the case $Inv \cup Xpos \neq \emptyset$ and $\varepsilon \in Dup$ follows by analogous arguments and is left to the reader. □

3.2.7 Adult Languages

Since evolutionary grammars can be considered as formal models for the evolution of genomes, the final stages of the development or evolution are of special interest, i.e. those genomes to which no mutation taken into consideration can be applied. In terms of languages we are interested in those words which do not allow a continuation of the derivation. Languages of such words are called adult languages and have been investigated extensively also in connection with L-systems, [113]. In this section we study adult languages of evolutionary grammars and show that they form the family recursively enumerable languages. We mention here that the adult languages of L-systems do not possess this rather surprising property.

Let $EG = (V, A, Del, Inv, Xpos, Dup)$ be an evolutionary grammar. Then the adult language of EG is defined by

$$A(EG) = \{w \mid w \in L(EG) \text{ and there is no } w' \text{ with } w \Longrightarrow w'\}.$$

The language $L(EG)$ contains all words which can be generated by iterated derivation steps, and the adult language $A(EG)$ contains those words from the generated language $L(EG)$ which do not allow further derivation steps. Therefore a word $w \in L(EG)$ belongs to the adult language if w does not contain a subword which can be deleted or reversed or translocated or duplicated. This implies the following statement.

Lemma 3.2.5 *Let $EG = (V, Del, Inv, Xpos, Dup, A)$ be a context-free evolutionary grammar. Then*

$$A(EG) = L(EG) \cap (V^* \setminus V^*(Del \cup Inv \cup Xpos \cup Dup)V^*).$$

We now present a characterizations of the set of recursively enumerable languages by adult languages of evolutionary grammars.

Theorem 3.2.15 *The family of adult languages of context-free evolutionary grammars coincides with the family of recursively enumerable languages.*

Proof. By Theorem 3.2.4 and Lemma 3.2.5 any adult language of a context-free evolutionary grammar is recursively enumerable.

Now let L be an arbitrary recursively enumerable language. We construct an evolutionary grammar EG' such that $A(EG') = L$.

In order to get $EG' = (V', A', Del', Inv', Xpos', Dup)$ we modify the grammar $EG = (V, A, Del, Inv, Xpos, Dup)$ with $L = L(EG) \cup T^*$ given in the proof of Theorem 3.2.4 as follows: We obtain V' by adding a new symbol \$ to the alphabet V. Then we define the morphisms $h : V \to V'$ by $h(A) = A\$$ for $A \in N$ and $h(x) = x$ for $x \in V \setminus N$. Then we set

$$\begin{aligned}
A' &= h(A)\,, \\
Xpos' &= h(Xpos)\,, \\
Dup' &= h(Dup) \cup \{\lhd, \$\}\,, \\
Del' &= \{A\$B\$C\$, \dashv\} \cup \{\perp_i \$S \lhd \ \rhd \mid 1 \le i \le n\} \cup \\
&\quad\ \{\lhd h(x_i) \perp_i \rhd \dashv \mid 1 \le i \le n\}\,, \\
Inv' &= \{\lhd h(x_i) \perp_i \mid 1 \le i \le n\} \cup \{S\$\perp_i mi(h(x_i)) \mid 1 \le i \le n\}\,.
\end{aligned}$$

Essentially, instead of a nonterminal A we use $A\$$ or $\$A$ (where the latter one only occurs since inversions can be involved). Thus the deletion of all symbols \$ in a word of $L(EG')$ gives a word of $L(EG)$, and conversely, for any word $w \in L(EG)$ there is a word $w' \in L(EG')$ such that the deletion of all \$ in w' gives w. This implies

$$L(EG) \cap T^* = L(EG') \cap T^* . \tag{3.4}$$

Now $L = A(EG')$ follows from the following remarks: For any word $w \in L(EG')$, by the construction

$$|w|_N \leq |w|_\$ \text{ and } |w|_{\rhd} = |w|_{\{\perp_i | 1 \leq i \leq n\}} \leq |w|_{\lhd} . \tag{3.5}$$

If $w \in L(EG')$ contains \$ or $\lhd$, then we can apply a duplication. If $w \in L(EG')$ contains $\dashv$, a deletion can be applied. Thus a word $w \in A(EG')$ cannot contain \$, $\lhd$ and $\dashv$. By (3.5) w cannot contain $\rhd$, $\perp_i$, $1 \leq i \leq n$, and elements of N. Hence $w \in A(EG')$ contains only terminals. On the other hand, any word $w \in L(EG') \cap T^*$ belongs to $A(EG')$ since any derivation step requires the presence of symbols not belonging to T. Therefore we obtain from (3.4)

$$A(EG') = L(EG') \cap T^* = L(EG) \cap T^* = L .$$

□

From the proof of Theorem 3.2.15 we see that adult languages can be obtained as the intersection of the set of all generated words with an monoid. This corresponds to the situation well-known for reduced[1] context-free grammars G: On one side $L(G)$ is the intersection of the set of all generated words with the set of terminal words, and on the other side $L(G)$ consists of all words which allow no further derivation.

Another Version of Adult Languages

In the theory of Lindenmayer systems another definition of adult languages is used (see [113], pages 70–78 and 287) . By definition, in a Lindenmayer system at every moment a derivation step can be performed, and thus the adult language consists of all words which are not changed by derivation steps. For an evolutionary grammar $EG =$

[1]For any nonterminal A, there is a rule with the left-hand side A.

$(V, A, Del, Inv, Xpos, Dup)$, the modified adult language $mA(EG)$ is defined as the set of all words $w \in L(EG)$ such that either $w \in A(EG)$ or $w \in L(EG)$ and $w \Longrightarrow z$ implies $z = w$.

Let $EG = (V, A, Del, Inv, Xpos, Dup)$ be an evolutionary grammar and let w be a word generated by EG such that $z = w$ holds for any z with $w \Longrightarrow z$.

Obviously, if there is an element $u \neq \varepsilon$ from Del and Dup, respectively, which can be applied to w, then there is a $z \neq w$ with $w \Longrightarrow z$. Thus the only applicable element from Del and Dup is ε, and the cancellation or duplication of ε in a word does not change the word.

Moreover, if we can apply an inversion $u \in Inv$ to w yielding z, then $w = w_1 u w_2$ and $z = w_1 mi(u) w_2$. Now $w = z$ implies $u = mi(u)$, and thus the application of the inversion u to an arbitrary word does not change the word.

Furthermore, let u be a transposition from $Xpos$ such that its application to w does not change w. If we shift u by one letter to the right, we obtain $w = v_1 u a v_2$ and $z = v_1 a u v_2$. $w = z$ implies $ua = au$. By the famous theorem by Lyndon and Schützenberger (see Lemma 1.7 in [125]) we get $u \in \{a\}^*$. Shifting to other positions we can show that $w \in \{a\}^*$. Thus the application of the transposition u to any word of $v \in \{a\}^*$ with $|v| \geq |u|$ does not change v.

We now consider the evolutionary grammar

$$EG_1 = (V, A, Del \setminus \{\varepsilon\}, Inv \setminus \{u \mid u = mi(u)\}, Xpos, Dup \setminus \{\varepsilon\}).$$

Obviously, any word $w \in mA(EG) \setminus A(EG)$ to which elements of Del or Dup or Inv can be applied without changing w is contained in $A(EG_1)$. Thus we obtain

$$mA(EG) = A(EG_1) \cup \bigcup_{a \in V} \{w \mid w \in L(EG) \cap \{a\}^*, s(a) \leq |w| \leq t(a)\} \tag{3.6}$$

where

$$s(a) = min\{|u| \mid u \in T \cap \{a\}^*$$

and

$$t(a) = \begin{cases} min\{|u| \mid u \in (Dup \cup Del) \cap \{a\}^+, \\ \text{ if } (Dup \cup Del) \cap \{a\}^+ \neq \emptyset \\ \\ \infty, \text{ otherwise.} \end{cases}$$

3.3 Duplication Grammars

String duplications or duplications of segments of strings are rather frequent in both natural and genetic languages. For motivations coming from linguistics, we refer to [86] and [110].

We consider here the context-free variants of duplication grammars. We investigate their generative capacity, their mutual relationship, and their relationship to the context-sensitive duplication grammars.

Based on [32], Martin-Vidè and Păun introduced in [89] a generative mechanism (similar to the one considered in [33]) based only on duplication: one starts with a given finite set of strings and produces new strings by copying specified substrings to certain places in a string, according to a finite set of duplication rules. This mechanism is studied in [89] from the generative power point of view.

The section considers the context-free versions of duplication grammars - this formalizes a possible hypothesis that duplications appear more or less at random within the genome in the course of its evolution. We follow [94] where some problems left open in [89] were solved, new results concerning the generative power of context-sensitive and context-free duplication grammars were proved, and the two classes of grammars were compared. Finally, some decision problems are discussed.

A *context-sensitive duplication rule* is a triple whose components are strings over a given alphabet (in the case of DNA the alphabet consists of the four nucleotids), say (u, x, v), which has the following interpretation:

- the string x, which appears to the left of uv in the processed string, is inserted in between u and v;

- the string x, which appears to the right of uv in the processed string, is inserted in between u and v;
- the string x which appears in between u and v is doubled.

A *context-free duplication rule* is a string over the given alphabet, say x, whose effect is the duplication of x either to the right of, or to the left of, or immediately after, an already existing copy of x. Clearly, context-free duplication rules may be viewed as context sensitive duplication rules whose contexts are empty.

In vivo, cross-over takes place just between homologous chromosomes (chromosomes of the same type and of the same length), see [57]. A model of a cross-over between a DNA molecule and its replicated version is considered in in the next section - this is a model for a cross-over between "sister" chromatides. One specifies an initial finite set of strings and a finite set of cross-over rules of the form $(\alpha, \beta, \gamma, \delta)$. It is assumed that every initial string is replicated so that two identical copies of every initial string are available. The first copy is cut between the segments α and β and the other one is cut between γ and δ. Now, the last segment of the second string gets attached to the first segment of the first string, and a new string is obtained. More generally, another string is also generated, by linking the first segment of the second string with the last segment of the first string. Iterating the procedure, one gets a language.

The main idea of this approach is schematically presented in the Figure 3.1.

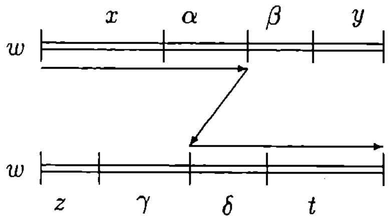

Figure 3.1.

Hence, the splicing operation introduced by T. Head, see, e.g., [60] is performed here between identical strings. It is easily seen that one obtains the insertion of a substring of w in w; this induces a duplication of some chromosomes into genome. This type of recombination is considered to be the main way of producing tandem repeats or block deletions in chromosomes.

A *(context-sensitive) duplication grammar* is a construct

$$\Delta = (V, D_l, D_r, D_0, A),$$

where V is an alphabet, D_l, D_r, D_0 are finite subsets of $V^* \times V^+ \times V^*$, and A is a finite subset of V^+. The elements of D_l, D_r and D_0 are context-sensitive duplication rules, and elements of A are called axioms.

Given a duplication grammar as above and two words $x, y \in V^+$, we define the following three types of direct derivation relations in Δ:

$$\begin{aligned} x \ &\Longrightarrow_{D_l} \ y \text{ iff } x = x_1uvx_2zx_3,\ y = x_1uzvx_2zx_3, \\ &\quad \text{with } x_1, x_2, x_3 \in V^*, \text{ and } (u, z, v) \in D_l, \\ x \ &\Longrightarrow_{D_r} \ y \text{ iff } x = x_1zx_2uvx_3,\ y = x_1zx_2uzvx_3, \\ &\quad \text{with } x_1, x_2, x_3 \in V^*, \text{ and } (u, z, v) \in D_r, \\ x \ &\Longrightarrow_{D_0} \ y \text{ iff } x = x_1uzvx_2,\ y = x_1uzzvx_2, \\ &\quad \text{with } x_1, x_2, x_3 \in V^*, \text{ and } (u, z, v) \in D_0. \end{aligned}$$

The union of these relations is the direct derivation relation of Δ, denoted by $\Longrightarrow$, and the reflexive and transitive closure of $\Longrightarrow$ is the derivation relation of Δ, denoted by $\Longrightarrow^*$. The language generated by the duplication grammar Δ is defined by

$$L(\Delta) = \{y \in V^* \mid x \Longrightarrow^* y, \text{ for some } x \in A\}.$$

Thus, the language of Δ consists of all words obtained by beginning with strings in A, and applying iteratively duplication rules from $D_l \cup D_r \cup D_0$. The application of a rule to a string means to copy one of its substrings to the left of, or to the right of, or next to its "given" occurrence. Because each of the three sets of rules may be empty, one obtains seven families of languages denoted by $DUPL(X)$, $X \in \{l, r, 0, lr, l0, r0, lr0\}$; the presence of a letter within X means that the corresponding set of rules is non-empty, e.g., for $X = l0$, $D_l \neq \emptyset$, $D_0 \neq \emptyset$ and $D_r = \emptyset$.

Analogously, we define a context-free duplication grammar as a construct $\Delta = (V, D_l,$
$D_r, D_0, A)$, where V and A have the same interpretation as above, but D_l, D_r, D_0 are finite subsets of V^+ whose elements are context-free duplication rules. Given a context-free duplication grammar as above and two words $x, y \in V^+$, we define three types of direct derivation relations:

$$\begin{aligned}
x &\models_{D_l} y \text{ iff } x = x_1x_2zx_3,\ y = x_1zx_2zx_3,\\
&\qquad \text{with } x_1, x_2, x_3 \in V^*, \text{ and } z \in D_l,\\
x &\models_{D_r} y \text{ iff } x = x_1zx_2x_3,\ y = x_1zx_2zx_3,\\
&\qquad \text{with } x_1, x_2, x_3 \in V^*, \text{ and } z \in D_r,\\
x &\models_{D_0} y \text{ iff } x = x_1zx_2,\ y = x_1zzx_2,\\
&\qquad \text{with } x_1, x_2, x_3 \in V^*, \text{ and } z \in D_0.
\end{aligned}$$

Again, the union of these relations is the direct derivation relation, denoted by $\models$, and the reflexive and transitive closure of $\models$ is the derivation relation, denoted by $\models^*$. The language generated by the context-free duplication grammar Δ is defined by

$$L(\Delta) = \{y \in V^* \mid x \models^* y, \text{ for some } x \in A\}.$$

Again, we get seven families of languages denoted by $CFDUPL(X)$, $X \in \{l, r, 0, lr, l0,$
$r0, lr0\}$.

3.3.1 A Short Comparison

We begin by settling the relationships among the seven families of context-free duplication languages.

Theorem 3.3.1 *The relations in the following diagram hold, where an arrow indicates a strict inclusion and a dotted line links two incomparable families.*

$$\begin{tikzcd}
CFDUPL(l) \arrow[drr] \arrow[ddrrrr] \arrow[ddr, dash, dashed] \arrow[dddd, dash, dashed] & & & & \\
& & CFDUPL(l0) \arrow[dr] & & \\
& CFDUPL(0) \arrow[rr] & & CFDUPL(lr0) & CFDUPL(lr) \arrow[l] \\
& & CFDUPL(r0) \arrow[ur] & & \\
CFDUPL(r) \arrow[urr] \arrow[uurrrr] \arrow[uur, dash, dashed] & & & &
\end{tikzcd}$$

Figure 3.2.

Proof. The language $\{a^n b^m a^p b^q | n, m, p, q \geq 1\}$ is in $CFDUPL(0)$ (one starts with $abab$ and doubles either an occurrence of a or an occurrence of b) but not in $CFDUPL(lr)$. To see the latter, we note that each context-free duplication grammar having just left and right duplication rules generates strings in $a^+b^+a^+b^+a^+b^+$; a contradiction.
By a similar reasoning, the language $\{a^n b^m | n, m \geq 1\}$ belongs to

$$CFDUPL(l0) \cap CFDUPL(r0) \cap CFDUPL(lr)$$

but not to

$$CFDUPL(l) \cup CFDUPL(r).$$

The language $\{a, b, c\}^+$ is in $CFDUPL(r) \cap CFDUPL(l)$ (the initial set contains all strings of length at most 3, each letter a, b, c appearing at most once; duplication rules allow copying of any letter to the right/left of one of its occurrences.) Because there are arbitrarily long square-free strings in $\{a, b, c\}^+$, [128], it follows that $\{a, b, c\}^+ \notin CFDUPL(0)$.

Finally,

$$\{a, b, c\}^+\{\$\}^+\{d, e, f\}^+ \in CFDUPL(lr0) \setminus$$
$$(CFDUPL(l0) \cup CFDUPL(r0))$$

which conclude the proof. □

The following result concerning the relationships among the context-sensitive families of duplication languages has been proved in [89].

Theorem 3.3.2.[89]
1. *The families $DUPL(l)$ and $DUPL(r)$ are incomparable.*
2. *The following inclusions*

$$DUPL(r) \cup DUPL(l) \subset DUPL(lr)$$
$$DUPL(0) \subset (DUPL(r0) \cap DUPL(l0))$$

are proper.

It is an *open problem* whether or not $DUPL(0)$ is included in $DUPL(l)$ or in $DUPL(r)$. However, we have

Proposition 3.3.1 *CFDUPL(0) is strictly included in DUPL(lr).*

Proof. Let $\Delta = (V, \emptyset, \emptyset, D_0, A)$ be a duplication grammar with $D_0 = \{x_1, x_2, \ldots, x_n\}$. Construct a duplication grammar $\Delta' = (V, D_l, D_r, \emptyset, A')$, where

$$\begin{aligned} D_l = D_l &= \{(x_i, x_i, \varepsilon) | 1 \le i \le n\}, \\ A' &= \{z \in L(\Delta) | \text{ each } x_i \text{ has at most two non-overlapped} \\ &\quad \text{occurrences in } z\}. \end{aligned}$$

It is easy to see that A' is a finite set, and $L(\Delta) = L(\Delta')$. □

Along the same lines, we have

Theorem 3.3.3 *$CFDUPL(X) \subset DUPL(X)$, for all $X \in \{0, l, r, l0, r0, lr, lr0\}$.*

Proof. It suffices to provide languages that prove all inclusions to be strict. The duplication grammar $\Delta = (\{a, b\}, \{(\varepsilon, a, a), (\varepsilon, b, b)\}, \emptyset, \emptyset, \{ab, a^2b, ab^2, a^2b^2\}$ generates $L_1 = \{a^n b^m | n, m \ge 1\}$. Hence L_1 is in $DUPL(l)$ (also in $DUPL(r)$) but not in $(CFDUPL(l) \cup CFDUPL(r)$.

Similarly, $\{a^n b^m a^p b^q | n, m, p, q \ge 1\} \in DUPL(lr) \setminus CFDUPL(lr)$.

One can show that $\{a^n b^n ab | n \geq 1\}$ cannot be generated by any context-free duplication grammar. On the other hand, $\{a^n b^n ab | n \geq 1\} \in DUPL(lr0)$ (see [89]).

Take now the language $L_2 = \{ab^n c^m d^p e | 1 \leq n, m, p \leq 3\}^+$. This language can be obtained by starting with the string $abcde$ and iteratively applying rules from the set

$$\begin{aligned} D_0 &= \{(\varepsilon, abcde, \varepsilon), (a, b, c), (ab, b, c), (b, c, d), (bc, c, d), \\ &\quad (c, d, e), (cd, d, e)\}. \end{aligned}$$

Consider the homomorphism $h : \{a, b, c\}^* \longrightarrow \{a, b, c, d, e\}^*$ defined by $h(a) = ab^3cde$, $h(b) = abc^3de$, $h(c) = abcd^3e$. Let x be an arbitrarily long square-free string over $\{a, b, c\}$. The string $h(x)$ is in L_2. It is easy to notice that the adjacent identical substrings in $h(x)$ are only the letters from $\{a, b, c\}$. If L_2 were in $CFDUPL(0)$, then any context-free duplication grammar generating L_2 would generate strings containing arbitrarily many adjacent occurrences of the same letter from $\{a, b, c\}$; a contradiction. □

3.3.2 Observations on the Generative Power

We start by considering unary alphabets. We will prove that in this case the generative power of duplication grammars equals the accepting power of deterministic finite automata. To this end, we prove the following lemma.

Lemma 3.3.1 *Over the unary alphabet, the equality $DUPL(X) = CFDUPL(0)$ holds for any $X \in \{l, r, 0, lr, l0, r0, lr0\}$.*

Proof. Let $\Delta = (\{a\}, D_l, D_r, D_0, A)$ be a duplication grammar. Let

$$\begin{aligned} D_l &= \{(u_l, a^{i_l}, v_l) | 1 \leq l \leq n\}, \\ D_r &= \{(x_l, a^{j_l}, y_l) | 1 \leq l \leq m\}, \\ D_0 &= \{(z_l, a^{k_l}, w_l) | 1 \leq l \leq p\}. \end{aligned}$$

Take

$$\alpha = max(\{|uxv| \ : \ (u, x, v) \in D_l \cup D_r \cup D_0\} \cup \{|x| \ : \ x \in A\}.$$

Consider now the context-free duplication grammar

$$\Delta' = (\{a\}, \emptyset, \emptyset, D_0', A'),$$

where

$$\begin{aligned} A' &= \{x | x \in L(\Delta), |x| \leq 3\alpha\}, \\ D_0' &= \{a^q | q = \sum_{s=1}^{n} \alpha_s i_s + \sum_{s=1}^{m} \beta_s j_s + \sum_{s=1}^{p} \gamma_s k_s,\ \alpha \leq q \leq 2\alpha\}. \end{aligned}$$

We claim that $L(\Delta) = L(\Delta')$. Note that each rule in D_0' is applicable to strings of length at least α. Furthermore, each application of a rule in D_0' simulates the application of a sequence of rules from $D_l \cup D_r \cup D_0$. Consequently, $L(\Delta') \subseteq L(\Delta)$.

All strings of length at most 3α from $L(\Delta)$ are also in $L(\Delta')$. Let z be the shortest string in $L(\Delta)$ such that $|z| > 3\alpha$. Then there exists a derivation in Δ':

$$x \Longrightarrow^+ y \Longrightarrow^+ z$$

with

$$\begin{aligned} &(i) && x \in A, \\ &(ii) && \alpha \leq |y| \leq 3\alpha, \\ &(iii) && \alpha \leq |z| - |y| \leq 2\alpha. \end{aligned}$$

Because $y \in A'$ one may write $y \Longrightarrow_{D_0'} z$, and so $z \in L(\Delta')$. Inductively, $L(\Delta) \subseteq L(\Delta')$. □

Theorem 3.3.4 *A language over a unary alphabet is regular if and only if it is generated by a duplication grammar.*

Proof. By the previous lemma, it suffices to consider duplication grammars with just context-free duplication rules whose effect is to double an occurrence of a substring. Let $L \subseteq \{a\}^*$ be a regular language. Then, there exist a finite set F and the positive integers $k_i, 1 \leq i \leq m$, and $q > max\{|x| \mid x \in F\}$ such that

$$L = F \cup \bigcup_{i=1}^{m} \{a^{k_i + nq} | n \geq 1\}$$

This can be easily seen if one considers a deterministic finite automaton accepting L, for which the transition function is defined everywhere.

Consider now the duplication grammar:

$$\Delta = (\{a\}, \emptyset, \emptyset, \{a^q\}, F \cup \{a^{k_i+q} | 1 \leq i \leq m\}.$$

Clearly, $L = L(\Delta)$. Duplications can never be carried out on words of F.

Conversely, let us consider a duplication grammar $\Delta = (\{a\}, \emptyset, \emptyset, D_0, A)$, with $D_0 = \{a^{c_1}, a^{c_2}, \ldots, a^{c_n}\}$. Let

$$p = gcd(c_1, c_2, \ldots c_n),$$

where *gcd* means the greatest common divisor. If $L(\Delta)$ is finite, then it is obviously regular. If $L(\Delta)$ is an infinite set, then there are $t_i, 1 \leq i \leq s,\ s \leq p$, such that

$$L(\Delta) = F \cup \bigcup_{i=1}^{s} \{a^{t_i+kp} | k \geq 0\},$$

for some finite set F. Consequently, $L(\Delta)$ is regular which completes the proof. □

The next result settles a problem left open in [89].

Theorem 3.3.5 *All regular languages are in $DUPL(X)$, $X \in \{l, r, l0, r0, lr, lr0\}$.*

Proof. We present a proof for $DUPL(r)$, the proofs for other cases are analogous. Let R be a regular language recognized by the deterministic finite automaton $M = (Q, V, \delta, q_0, F)$ with the total transition function δ. Let for each state q, C_q be defined as follows:

$$C_q = \{x \in V^+ | \delta(q, x) = q \text{ by passing each state, different from } q, \text{ at most once}\}.$$

For strings $x, y \in V^*$, we define the equivalence relation $\sim_R$ as follows:

$$(x \sim_R y) \text{ iff } (uxv \in R \text{ iff } uyv \in R), \text{ for any } u, v \in V^*.$$

It is well-known (see e.g. [114]) that $V^*/\sim_R$ (the quotient of V^* by $\sim_R$) is finite; let k be the cardinality of $V^*/\sim_R$ (the index of $\sim_R$).

Now, one constructs the duplication grammar $\Delta = (V, \emptyset, D_r, \emptyset, A)$, where

$$\begin{aligned} D_r &= \bigcup_{q\in Q, C_q\neq\emptyset} \{(x,y,\varepsilon) | xy \sim_R x,\ |x| < k, y \in C_q\}, \text{ and} \\ A &= \{w \in R | \text{ for each } q \in Q, \text{ each string in } C_q \\ &\qquad \text{has at most } k \text{ non-overlapping occurrences in } w\}. \end{aligned}$$

We claim that A is finite. Indeed, no word longer than $(k+1)l \cdot card(Q)$, where $l = max\{card(C_q) | q \in Q\}$, is in A. To see this, assume that such a word, say w, is in A; so $|w| = p \geq (k+1)l \cdot card(Q)$. Let $q_0, q_1, \ldots, q_p$, $q_p \in F$, be the sequence of states that accepts w. At least $(k+1)l$ states in this sequence must be the same; assume that q is such a state. But then w contains at least $k+1$ identical substrings in C_q; a contradiction.

Clearly, $L(\Delta) \subseteq R$. Let z be the shortest word in $R \setminus L(\Delta)$. Thus, there exists $x \in C_q$, for some $q \in Q$, such that x occurs more than k times in z. Let $z = wxy$, with $|w| \geq k$, where the given occurrence of x is the last (rightmost) occurrence of x in z. Let $z = uvxy$ with $|v| = k$. Thus v has $k+1$ prefixes, and so there are two prefixes v_1, v_2 of v such that $v_1 \sim_R v_2$ and $|v_1| < |v_2|$. We choose the closest pair of such prefixes. By replacing v_2 by v_1 in v we get a string $uv'xy$ which is in $L(\Delta)$ because it is in R and it is shorter than z. Moreover, $v_2 = v_1 t$, where t must be in C_q, for some $q \in Q$ (because of the choice of v_1 and v_2). Consequently, $(v_1, t, \varepsilon) \in D_r$, and so $uv'xy \Longrightarrow_{D_r} z$. Thus $z \in L(\Delta)$; a contradiction.

Analogously one proves that each regular language is in $DUPL(l)$. □

We recall that the family $DUPL(0)$ is incomparable with the family of regular languages.

The position of the class of regular languages with respect to the classes of context-free duplication languages is given by the next theorem.

Theorem 3.3.6 *The family of regular languages is incomparable with any of the families $CFDUPL(X), X \neq 0$.*

Proof. The regular language $V^+\{c\}^+V^+$, where V contains at least three symbols and $c \notin V$, cannot be generated by any context-free duplication grammar. Indeed, if a context-free duplication grammar generates all strings in $V^+\{c\}^+V^+$, then it must contain left/right duplication rules involving strings in $V^+ + c^+$. Therefore, also strings in $V^+\{c\}^+V^+\{c\}^+V^+$ can be generated.

Consider now the Dyck language over $\{a, b\}$, denoted by D_{ab}, and the non-regular language $L = \{ab\}D_{ab}$. This language is in $CFDUPL(r)$. The context-free duplication grammar $\Delta = (\{a, b\}, \emptyset, \{ab\}, \emptyset, \{abab\})$ with only right duplication rules generates L. Clearly, $L(\Delta) \subseteq L$; let z be the shortest string in $L \setminus L(\Delta)$. If $z = abxy$, with $x, y \in D_{ab}$, then ab yields z in Δ as follows:

$$ab \models^* aby \models^* abxy.$$

If $z = abaxb$, with $x \in D_{ab}$, then the derivation $ab \models abab \models^* abaxb$ is possible in Δ. Consequently, $L(\Delta) = L$. □

The relation between $CFDUPL(0)$ and the class of regular languages remains open.

Recall that a homomorphism which erases some symbols and leaves the others symbols unchanged is called a *projection*. A projection $h : (V \cup V')^* \longrightarrow V^*$ that erases the symbols in V' only is the projection of V, denoted by pr_V.

Theorem 3.3.7 *For each context-free language $L \in V^*$, there exists a language L in $CFDUPL(r)$ ($CFDUPL(l)$) and a homomorphism h such that $L = pr_V(h^{-1}(L'))$.*

Proof. Let $G = (N, V, S, P)$ be a context-free grammar generating L. Assume that

$$P = \bigcup_{i=1}^{n} \{A_i \longrightarrow x_{i,j} | 1 \leq j \leq r_i\},$$

with $S = A_1$. Furthermore, we assume that $\varepsilon \notin L$. Let $V' = N \cup V \cup \{c_i | 1 \leq i \leq n\} \cup \{d\}$, where c_i, d, are new symbols. Let then Δ be the duplication grammar $(V', \emptyset, D_r, \emptyset, A)$, where

$$\begin{aligned} D_r &= \{(c_i x_{i,j} | 1 \leq i \leq n,\ 1 \leq j \leq r_i\}, \text{ and} \\ A &= \{c x_{1,1} d c x_{1,2} d \ldots d c x_{1,r_1} d c x_{2,1} d \ldots d c x_{n,r_n} d A_1\}. \end{aligned}$$

Now, let h be the homomorphism

$$h : (V \cup \{[i,j] | 1 \leq i \leq n, 1 \leq j \leq r_i\} \cup \{c_i | 1 \leq i \leq n\} \longrightarrow (V')^*$$

such that

$$\begin{aligned} h([i,j]) &= c_i x_{i,j} d, 1 \leq i \leq n,\ 1 \leq j \leq r_i, \\ h(c_i) &= A_i c_i, 1 \leq i \leq n, \text{ and} \\ h(a) &= a,\ a \in V. \end{aligned}$$

It is easy to see that $pr_V(h^{-1}(L(\Delta))) = Gen(G)$. Clearly, whenever a substring $c_i x_{i,j}$ is copied, this is done somewhere to the right of the last occurrence of d - otherwise one gets a string "rejected" by applying the inverse homomorphism h. Also, all strings that contain nonterminal occurrences that are not immediately followed by some c_i, to the right of the last occurrence of d, are rejected in the same way. Moreover, every occurrence of a nonterminal A_i, situated to the right of the last occurrence of d, has to be followed by just one occurrence of c_i. In this way duplication rules simulate the application of production rules in G. □

3.3.3 Decision Problems

We discuss in this section some basic decision problems. We begin by pointing out that the "totality problem" is decidable for all families of duplication languages.

Theorem 3.3.8 *Let Δ be a duplication grammar over the alphabet V. It is decidable whether or not $L(\Delta) = V^*$.*

Proof. We will consider duplication grammars having only left duplication rules - the other types of duplication grammars can be treated in a similar way. Let $\Delta = (V, D_l, \emptyset, \emptyset, A)$ be a duplication grammar. The main point of our argument is the following property

$$L(\Delta) = V^* \text{ if and only if } \{x \in V^* : |x| \leq k+1\} \subset L(\Delta),$$

where $k = max\{|x| : x \in A\}$.

The "only if" part is obvious. For the "if" part of the proof, assume that z is a shortest word in $V^* \setminus L(\Delta)$. This word can be written as $z = ya$ with $a \in V$. Hence $y \in L(\Delta) \setminus A$, and so there exists $x \in A$ such that $x \Longrightarrow^+_{D_r} y$. Because $|xa| < |ya|$, it follows that $xa \in L(\Delta)$. But, also $xa \Longrightarrow^+_{D_r} ya = z$. To conclude, it suffices to note that the inclusion $\{x \in V^* : |x| \leq k+1\} \subset L(\Delta)$ is decidable due to the decidabilty of the membership problem. □

It is proved in [89] that the membership of a context-free language in the family of languages $DUPL(X), X \neq 0$, is not decidable. Our next theorem extends this result to the families of context-free duplication languages, as well as to $DUPL(0)$.

Theorem 3.3.9 *It is not decidable whether or not a context-free language is in a family $CFDUPL(X), X \notin \{r, l\}$.*

Proof. The proof is similar to the one in [89]. Let G be an arbitrary context-free grammar with the terminal alphabet $\{a, b\}$, and let

$$L = Gen(G)\{c, d\}^* \cup \{a, b\}^*\{c^n d^n | n \geq 1\}.$$

If $Gen(G) = \{a, b\}^*$, then $L = \{a, b\}^*\{c, d\}^*$ which is in $CFDUPL(X)$, for all $X \notin \{r, l\}$. It is easily seen that the grammar $\Delta = (\{a, b, c, d\}, \emptyset, \emptyset, D_0, A)$, with

$$D_0 = \{a, b, c, d, ab, ba, cd, dc\},$$

and

$$A = \{a, b, c, d, ab, aba, ba, bab, cd, cdc, dc, dcd\},$$

generates $\{a, b\}^*\{c, d\}^*$. The reader my easily check this assertion.

If $Gen(G) \neq \{a, b\}^*$, then L cannot be generated by any context sensitive duplication grammar (see the proof of Theorem 4 in [89]). Consequently, $L \in CFDUPL(X)$ for $X \notin \{r, l\}$, if and only if $Gen(G) = \{a, b\}^*$, which is undecidable. □

This result can be also extended to the families $CFDUPL(r)$ and $CFDUPL(l)$.

Theorem 3.3.10 *It is not decidable whether or not a context-free language is in a family $CFDUPL(X), X \in \{r, l\}$.*

Proof. The proof is based on a reduction to the Post Correspondence Problem (PCP). Take an arbitrary instance of PCP, i.e., two arbitrary n-tuples of nonempty strings over the alphabet $\{a, b\}$:

$$x = (x_1, x_2, \ldots, x_n),$$

$$y = (y_1, y_2, \ldots, y_n).$$

Then, consider the languages

$$L_z = \{ba^{t_1}ba^{t_2}\ldots ba^{t_k}cz_{t_k}\ldots z_{t_2}z_{t_1} | k \geq 1\} \text{ for } z \in \{x, y\},$$

$$L_s = \{w_1cw_2cmi(w_2)cmi(w_1) | w_1, w_2 \in \{a, b\}^*\}, \text{ and}$$

$$L(x, y) = \{a, b, c\}^* - (L_x\{c\}mi(L_y) \cap L_s).$$

It is known that $L(x, y)$ is a context-free language. For every solution $(i_1, i_2, \ldots, i_k)$ of $PCP(x, y)$ the strings

$$ba^{i_1}ba^{i_2}\ldots ba^{i_k}cx_{i_k}\ldots x_{i_2}x_{i_1}cmi(y_{i_1})mi(y_{i_2})\ldots$$
$$mi(y_{i_k})ca^{i_k}b\ldots ba^{i_2}ba^{i_1}b$$

are not in $L(x, y)$.
Clearly, when $L(x, y) = \{a, b, c\}^*$, then $L(x, y)$ is in $CFDUPL(r) \cap CFDUPL(l)$.

Now, it is sufficient to prove that $L(x, y) \notin CFDUPL(l) \cup CFDUPL(r)$ if $L(x, y) \neq \{a, b, c\}^*$.
Let us suppose that $L(x, y) = L(\Delta)$, $\Delta = (\{a, b, c\}, \emptyset, D_r, \emptyset, A)$. We choose a solution $(i_1, i_2, \ldots i_k)$ such that

$$|x_{i_k}x_{i_{k-1}}\ldots x_{i_1}| > max\{|w| \, |w \in A\}.$$

For $\{a, b\}^* \subseteq L(\Delta)$, there exists a word $w \in A$ such that

$$w \models^* mi(y_{i_1})mi(y_{i_2})\ldots mi(y_{i_k}) \in L(\Delta).$$

By the choice of the solution $(i_1, i_2, \ldots, i_k)$ the word

$$z = ba^{i_1}ba^{i_2}\ldots ba^{i_k}cx_{i_k}\ldots x_{i_2}x_{i_1}cwca^{i_k}b\ldots ba^{i_2}ba^{i_1}b$$

is in $L(\Delta)$.

Therefore, we get

$$z \models^* ba^{i_1}ba^{i_2}\dots ba^{i_k}cx_{i_k}\dots x_{i_2}x_{i_1}cmi(y_{i_1})mi(y_{i_2})\dots$$
$$mi(y_{i_k})ca^{i_k}b\dots ba^{i_2}ba^{i_1}b,$$

a contradiction. Hence the theorem holds. □

Finally, we consider "nonemptiness of the intersection problem" for $DUPL(X)$, $X \neq 0$.

Theorem 3.3.11 *It is undecidable whether or not $L_1 \cap L_2 = \emptyset$? for arbitrary two duplication languages in $DUPL(X), X \neq 0$.*

Proof. Let $x = (x_1, x_2, \dots, x_n), y = (y_1, y_2, \dots, y_n)$ be an instance of PCP, and let

$$L_x = \{w\$cd^{i_1}\$cd^{i_2}\dots\$cd^{i_k}x_{i_k}\dots x_{i_2}x_{i_1} | k \geq 1, 1 \leq i_j \leq n, 1 \leq j \leq k\}$$

$$\cup\{w\$cd^{i_1}\$cd^{i_2}\dots\$cd^{i_k}\$x_{i_k}\dots x_{i_2}x_{i_1} | k \geq 1, 1 \leq i_j \leq n, 1 \leq j \leq k\},$$

where $w = cdx_1cdy_1cd^2x_2cd^2y_2\dots cd^nx_ncd^ny_n$. L_y is defined analogously.

Clearly, the duplication grammar $\Delta = (\{a, b, c, d, \$, \#\}, \emptyset, D_r, \emptyset, \{w\$\#\})$, with

$$D_r = \{(\$, cd^ix_i, \#) | 1 \leq i \leq n\} \cup \{(\$, cd^ix_i, X) | 1 \leq i \leq n, X \in \{a, b\}\}$$

$$\cup\{(d, \$, a), (d, \$, b)\}$$

generates L_x.

This concludes the proof, because $L_x \cap L_y = \emptyset$ if and only if the instance (x, y) of PCP has no solution. □

3.4 Self Crossover Systems

In this paragraph we are dealing with a very particular case of crossover despite that this is not the only biologically significant case. One has asserted [57] that in vivo, crossover takes place just between homologous chromosomes (chromosomes of the same type and of the same length). A first attempt to model the homologous recombination was made in [65], where crossover between strings of equal length, which exchange each other segments of equal length, is proposed.

Roughly speaking, in the present paper, we try to model crossover between a DNA molecule and a its replicated version. Thus, our approach appears as a model for crossover between "sister" chromatids. In our opinion this restriction makes up a theoretical aspect of molecular biology that deserves to be investigated. What would happen if crossover occured only between a chromosome and its replica ?

The main idea of our approach is schematically presented in the figure below:

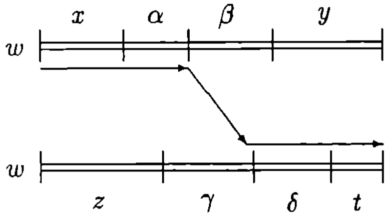

Figure 3.3.

One gives a starting finite set of string and a finite set of crossover rules $(\alpha, \beta, \gamma, \delta)$. One considers that every starting string is replicated so that, we have two identical copies for every initial string. The first copy is cut between the segments α and β and the other one is cut between γ and δ. Now, the last segment of the second string adheres to the first segment of the first string, and a new string is obtained. More generally, another string is also generated, by linking the first segment of the second string with the last segment of the first string. Iterating the procedure, we get a language.

We want to point out, at this moment, some connections between our approach and the other large scale operations in genome. If the situation is as in the Figure 3.3, then we have the deletion of a certain substring of w which may be viewed as the deletion of a segment of a chromosome.

If the situation is as in Figure 3.1 of the previous section, then we have the insertion of a substring of w in w; this appears as the duplication of a segment in a chromosome.

A self crossover system is a triple:

$$SCO = (V, A, R)$$

where V is an alphabet, A is a finite subset of V^*, and R is a finite commutative relation, $R \subset (V^* \times V^*)^2$. With respect to a self crossover system as above, for $x \in V^+$, we define:

$$x \bowtie y \quad \text{iff} \quad \begin{array}{ll} (i) & x = x_1\alpha\beta x_2 = x_3\gamma\delta x_4 \\ (ii) & y = x_1\alpha\delta x_4 \\ (iii) & (\alpha,\beta)R(\gamma,\delta). \end{array}$$

Note that $x \bowtie x_3\gamma\beta x_2$ follows from the definition of R. Moreover, the strings $x_1\alpha\delta x_4$ and $x_3\gamma\beta x_2$ are somehow "conjugated" namely, there exists $u \in V^*$ such that

$$\begin{array}{rcl} x & = & v_1uv_2, \text{ for some } v_1, v_2 \in V^*, \\ x_1\alpha\delta x_4 & = & v_1uuv_2, \\ x_3\gamma\beta x_2 & = & v_1v_2 \end{array}$$

Denote by $\bowtie^*$ the reflexive and the transitive closure of the relation $\bowtie$.

The language generated by a self crossover system as above is

$$L(SCO) = \{x \in V^* | w \bowtie^* x,\ w \in A\}$$

Example 3.4.1 *Take $V = \{a, b\}$, $A = \{bab\}$, and $R = \{(a, b; b, a), (b, a; a, b)\}$.*

We have

$$L(SCO) = \{bb\} \cup \{ba^{2^n}b | n \geq 0\}.$$

Indeed, $bab \bowtie ba^2b$ and $bab \bowtie bb$. Assuming that $bab \bowtie^* ba^{2^n}b$, by applying the rule $(a, b; b, a)$ to this string, we get $ba^{2^n}b \bowtie ba^{2^{n+1}}b$. By using the other splicing rule, we get $ba^{2^n}b \bowtie bb$. □

Example 3.4.2 *Consider* $V = \{a_1, a_2, \ldots, a_n\}$, $A = \{a_1 a_2 \ldots a_n\}$, $R = \{(\varepsilon, \varepsilon; \varepsilon, \varepsilon)\}$.

We state that $L(SCO) = V^*$. *We are going to prove our assertion by induction on* n. *It is obvious that the statement is true for* $n = 1$.

Let $z \in V^+$,

$$z = z_1 a_n^{t_1} z_2 a_n^{t_2} \ldots z_k a_n^{t_k} z_{k+1}$$

with $t_1, t_2, \ldots, t_k \geq 1, z_1, z_{k+1} \in (V - \{a_n\})^* z_1, z_2, \ldots, z_k \in (V - \{a_n\})^+$.

By the hypothesis of induction and following the crossover rule, we can perform the sequence of crossover below

$$w = a_1 a_2 \ldots a_n \bowtie a_1 a_2 \ldots a_n w \bowtie^* z_1 a_n w \bowtie z_1 a_n^2 w \bowtie^* z_1 a_n^{t_1} w$$

Going on, one obtains

$$z_1 a_n^{t_1} w \bowtie z_1 a_n^{t_1} ww \bowtie^* z_1 a_n^{t_1} z_2 a_n w \bowtie^* z_1 a_n^{t_1} z_2 a_n^{t_2} w \bowtie^* \ldots$$

$$\ldots \bowtie^* z_1 a_n^{t_1} z_2 a_n^{t_2} \ldots z_k a_n^{t_k} w \bowtie^* z_1 a_n^{t_1} z_2 a_n^{t_2} \ldots z_k a_n^{t_k} z_{k+1} a_n \bowtie z$$

Therefore, every string in V^* *can be generated by the above system, which concludes the proof.*

Denote by $\mathcal{L}(SCO)$ the family of languages generated by self crossover systems. The elements of $\mathcal{L}(SCO)$ will be referred as self crossover languages.

Theorem 3.4.1 *Every self cross-over language* L, *over* $\{a\}$, *is either a finite set or exist a finite set* $F \in V^*$ *and* $k > 0$ *such that* $L = F \cup \{a^n | n \geq k\}$.

Proof. Let $SCO = (\{a\}, A, R)$ be a self cross-over system. Since

$$L(SCO) = \bigcup_{x \in A} L(SCO_x),\ SCO_x = (\{a\}, x, R)$$

it suffices to show that any language $L(SCO_x)$ is either a singleton or of the form $\{a^n | n \geq k\}$, for some $k > 0$. Clearly, every language $L(SCO_x)$ is either a singleton or an infinite language.

Let $L(SCO_x)$ be an infinite language and

$$k = min\{|\alpha\delta| : (\alpha,\beta)R(\gamma,\delta)\}.$$

$$p = max(\{|\alpha| \,|(\alpha,\beta)R(\gamma,\delta)\} \cup \{|\beta| \,|(\alpha,\beta)R(\gamma,\delta)\})$$

Moreover, let $(\alpha,\beta;\gamma,\delta)$ be the cross-over rule which fulfils the minimal value of k. Of course, $L(SCO_x) \subseteq \{a^n | n \geq k\}$. We show that any $z = a^n, n \geq k$ is a string of $L(SCO_x)$. Because $L(SCO_x)$ is an infinite set, exists $m > n + p$ such that $a^m \in L(SCO_x)$.

Then, we have the two following decompositions

$$\begin{aligned} a^m &= a^{n-|\alpha|-|\delta|}a^{|\alpha|}/a^{m-n+|\delta|} \\ a^m &= a^{m-|\delta|}/a^{|\delta|}. \end{aligned}$$

where the cross-over sites are indicated by the symbol /, which result in generating of a^n. □

Lemma 3.4.1 *The language $L = a^*b^*a^*b^*$ cannot be generated by any self cross-over system.*

Proof. Assume that L can be generated by the system $SCO = (\{a,b\}, A, R)$. Let $z = a^nb^ma^kb^p$ be a string in L such that n, m, p, k are bigger than the length of the longest string over $\{a,b\}$ which occurs in a rule of R.

Please note that, at each cross-over between two identical strings of the above form, only the number of occurences of only one symbol, in just one of its two segments is modified. According to this remark, in order to get z, we have to produce a cross-over on a string of the form $a^nb^ma^kb^r, 0 < r < p$, in the sites indicated below by the symbol /:

$$a^nb^ma^kb^{r_1} \; / \; b^{r_2}, r_1 > 0$$

$$a^nb^ma^kb^{r_3} \; / \; b^{r_4}$$

But, we can choose also the following sites for cross-over:

$$a^nb^ma^kb^{r_1} \; / \; b^{r_2}$$

$$a^nb^{m_1} \; / \; b^{m_2}a^kb^r$$

and get the string $a^n b^m a^k b^{r_1+m_2} a^k b^r$, which leads to a contradiction. □

The next result is a consequence of the results got so far.

Theorem 3.4.2 *The family $\mathcal{L}(SCO)$ is incomparable with the families of regular and context-free languages, respectively.*

Theorem 3.4.3 *The families $\mathcal{L}(EG)$ and $\mathcal{L}(SCO)$ are incomparable.*

Proof. We prove that the self cross-over language $L = \{bb\} \cup \{ba^{2^n}b| \ n \geq 1\}$ cannot be generated by any evolutionary system. Assume the contrary and let $EG = (\{a,b\}, A, Del, Inv, Xpos, Dup)$ be an evolutionary grammar generating L. Since the set Dup has to be nonempty exists $a^k \in Dup$.

All strings $ba^{2^n}b$, $ba^{2^{n+1}}b$, $ba^{2^{n+2}}b$ are in $L(EG)$; consequently, by applying duplication rules to all strings $ba^{2^n}b$, $ba^{2^{n+1}}b$, $ba^{2^{n+2}}b$ we get strings in $L(EG)$. Therefore, there are integers $p < q < r$ such that

$$\begin{aligned} 2^n + k &= 2^p \\ 2^{n+1} + k &= 2^q \\ 2^{n+2} + k &= 2^r \end{aligned}$$

which leads to $2^{q+1} - 2^p = 2^r - 2^q$ or equivalently $2^p(2^{q+1-p} - 1) = 2^q(2^{r-q} - 1)$. It follows that $p = q$ that is contradictory.

Conversely, we observe that the language $L_1 = a^*b^*a^*b^*$ can be generated starting from $abab$ by iterating the duplication and deletion of both letters a and b, respectively. □

Now we shall prove that the family $\mathcal{L}(SCO)$ has very poor properties concerning the closure under usual operations in formal language theory.

Theorem 3.4.4 *The family $\mathcal{L}(SCO)$ is an anti-AFL and it is not closed under left/right derivatives and complement, too.*

Proof.

Union: The languages $L_1 = \{bb\} \cup \{ba^{2^n}b|n \geq 0\}$ and $L_2 = \{baaab\}$ are self cross-over languages but not their union. We omit the simple proof of this fact.

Catenation: The language $L = \{a^n b^m|n, m \geq 0\}$ can be generated by the self cross-over system (a detailed proof is left to the reader):

$$SCO = (\{a, b\}, \{ab\}, \{(a, \varepsilon; \varepsilon, a), (b, \varepsilon; \varepsilon, b), (\varepsilon, a; a, \varepsilon), (\varepsilon, b; b, \varepsilon)\}).$$

From Lemma 3.4.1 it follows that L^2 is not a self cross-over language.

Intersection with regular sets: Consider the intersection between the self cross-over language $\{a\}^*$ and the regular language $\{a^{2n}|n \geq 1\}$, which is not in $\mathcal{L}(SCO)$ due to Theorem 3.4.1.

Morphisms: Take the morphism $h : \{a, b\}^* \to \{a\}^*$ defined by $h(a) = h(b) = a$, and the self cross-over language $L = \{bb\} \cup \{ba^{2^n}b|n \geq 0\}$. However, $h(L) = \{aa\} \cup \{a^{2^n+2}|n \geq 0\}$ is not in $\mathcal{L}(SCO)$, as a consequence of Theorem 3.4.1.

Inverse morphisms: Take the self cross-over language $L = \{a^n|n \geq 3\}$ and the morphism $k : \{a, b\}^* \to \{a\}^*$ defined by $h(a) = a$, $h(b) = \varepsilon$. Clearly, $h^{-1}(L) = \{x \in \{a, b\}^*|\ |x|_a \geq 3\}$. We are going to prove that $h^{-1}(L) \notin \mathcal{L}(SCO$. Assume the contrary but notice that exists $k > 0$ such that at least a cross-over rule is applicable to the string $x = b^k ab^k ab^k ab^k$. Thus, x may be split

- between two a's,
- between an a and a b,
- between a b and an a.

Therefore, we should consider nine cases, but the reader can easily find out some appropiate sites such that each case leads to strings containing less than three a's, contradiction.

*Kleene operation *:* Consider the self cross-over language $L = \{bb\} \cup \{ba^{2^n}b|n \geq 0\}$ and assume that L^* can be generated by a self cross-over system. Following the same idea as for proving Lemma 3.4.1, we want to obtain the string

$$z = ba^{2^{n_1}}bba^{2^{n_2}}\dots bba^{2^{n_k}}b$$

with large enough $n_i, 1 \leq i \leq k$, pairwise different. For increasing the number of a's occurences in the last segment, we must split one copy of z somewhere on its last segment of a's. But, by choosing another segment, we will obtain a string which has a substring of the form ba^rb, and r is not a power of 2, contradiction.

Left derivatives: Take L the previous language. We shall prove that $\partial_b^l(L) = \{b\} \cup \{a^{2^n}b | n \geq 0\}$ is not in $\mathcal{L}(SCO)$.

Assumming the contrary, in order to get a string $a^{2^n}b$, with large enough n, we must apply a cross-over rule to a string, say $a^{2^m}b, n \neq m$. By crossing-over, the string $a^{2^m}b$ may give the strings $a^{2^m+k}b$ and $a^{2^m-k}b$, for some $k > 0$. For both strings must be in $\partial_b^l(L)$, it follows that exist $i \neq j$ such that $2^{m+1} = 2^i + 2^j$, contradiction.

The case of the right derivatives is symmetric.

Complement: For the non-closure under complement, take the language generates by the self cross-over system

$$SCO = (\{a, b\}, \{aaabbabb\}, R)$$

where

$$R = \{(\varepsilon, \varepsilon; x, y), (x, y; \varepsilon, \varepsilon) | x, y \in \{aa, ab, ba, bb\}\}$$

In order to prove that $L(SCO) = \{a, b\}^* - \{\varepsilon, a, b\}$ we establish the following two facts.

Fact 1. *All strings of length two over $\{a, b\}$ are in $L(SCO)$.*

This fact can be easily checked. For instance, the string ba can be obtained as follows:

$$aaabbabb \bowtie babb \bowtie ba$$

Fact 2. *Both strings $aaabbabba, aaabbabbb$ are in $L(SCO)$.*

The cross-over steps for generating the above strings are given below:

$$aaabbabb \bowtie aaabbabbabb \bowtie aaabbabba$$

and

$$aaabbabb \bowtie aaabbabbbabb \bowtie aaabbabbb$$

According to the first fact, we can assume, by induction, that all strings of length $n \geq 2$ over $\{a, b\}$ are in $L(SCO)$. Let z be a string of length $n+1$ over $\{a, b\}$ and $z = ua$. From the inductive hypothesis we have

$$aaabbabba \bowtie^* ua$$

hence, by combining with the second fact, $z \in L(SCO)$. Analogously, if $z = ub$.

Therefore, the complement of $L(SCO)$ is the language $\{\varepsilon, a, b\}$ which, obviously, is not in $\mathcal{L}SCO$. □

Theorem 3.4.5 *The family $\mathcal{L}(SCO)$ is not closed under duplications and deletions.*

Proof. If we try to duplicate the letter a in the strings from $\{bb\} \cup \{ba^{2^n}b | n \geq 0\}$, we get the language $\{ba^{2^n+1}b | n \geq 0\}$ which is not a self cross-over language. Indeed, let us assume that the string $ba^{2^n+1}b$ produces two conjugated strings, say $ba^{2^m+1}b$ and $ba^{2^k+1}b$. Observe that $2^{n+1} + 2 = 2^m + 2^k + 2$ holds that requires $n = m = k$. Consequently, our task leads to a finite set, contradiction.

The same reasoning is valid for deletions as well. □

Chapter 4

Other Operations

4.1 The PA-Matching Operation

We consider the PA-matching operation, used in DNA computing, as a formal operation on strings and languages. We investigate the closure of various families of languages under this operation, representations of recursively enumerable languages and decision problems. We also consider the dual operation of overlapping strings. All closure properties of families in the Chomksy hierarchy under both non-iterated and iterated PA-matching and overlapping operations are settled.

In the fastly emerging area of DNA computing, many new computability models are considered, where many of the operations used are inspired by the DNA behavior *in vivo* or *in vitro.* Examples of such operations are: the splicing operation (used in H systems), the annealing (used in sticker systems), and the insertion-deletion operations. These and other operations are discussed in [104].

Here we investigate yet another operation suggested by operations on DNA molecules, the so-called PA-matching operation, used in [109]. It is related to both the splicing and the annealing operations: starting from two single stranded molecules x, y, such that a suffix w of x is complementary to a prefix $\bar{w}$ of y, by annealing we can form the molecule with the double stranded part $\begin{pmatrix} w \\ \bar{w} \end{pmatrix}$ and the remaining sticky ends specified by x and y. The matching part is then ignored (removed), so that the resulting string consists of the prefix of x and the suffix of y which were not matched.

This operation is considered here as an abstract operation on formal languages. We relate it to other operations in formal language theory and we settle the closure properties of families in the Chomsky hierarchy under it. A dual operation is that of overlapping, where we keep a matching part of two strings. Also in this case we settle all closure properties of Chomsky families. Once again, it turns out that manipulation of DNA molecules leads to operations interesting from formal language theory point of view.

The set of all the proper prefixes and suffixes of the strings in a language $L \subseteq V^*$ are denoted by $PPref(L), PSuf(L)$, respectively.

For $L_1, L_2 \subseteq V^*$ we define the *left quotient* of L_1 with respect to L_2 by $L_2 \backslash L_1 = \{w \in V^* \mid xw \in L_1 \text{ for some } x \in L_2\}$. The *right quotient* is defined in the symmetric way. When L_2 is a singleton, $L_2 = \{x\}$, then we write $\partial^l_x(L)$ instead of $\{x\} \backslash L_1$ and this operation is called the *left derivative* of L_1 with respect to x. The *right derivative* is denoted by $\partial^r_x(L_1)$.

A *finite transducer* is a *gsm* which is able to change its current configuration without reading effectively the current input symbol. The finite transduction defined by a finite transducer M is denoted by T_M, similarly to the *gsm* mapping. If L is a regular language and M is a finite transducer, then $T_M(L)$ is also regular.

4.1.1 The Non-Iterated Case

The PA-matching operation consists of cutting two strings in two segments such that the prefix of one of them matches the suffix of another, removing these two matching pieces, and pasting the remaining parts.

Formally, given an alphabet V, a subset X of V^+, and two strings $u, v \in V^+$, one defines

$$PAm_X(u, v) = \{wz \mid u = wx, v = xz, \text{ for } x \in X, \text{ and } w, z \in V^*\}.$$

The operation is naturally extended to languages over V by

$$PAm_X(L_1, L_2) = \bigcup_{u \in L_1, v \in L_2} PAm_X(u, v).$$

When $L_1 = L_2 = L$ we write $PAm_X(L)$ instead of $PAm_X(L_1, L_2)$. Since we shall only deal either with finite sets X or with $X = V^+$, we use the notation $fPAm$ for finite PA-matching and the notation PAm for arbitrary PA-matching PAm_{V^+}.

The reader familiar with the splicing operation ([59], [60], [104]) may easily recognize a special variant of splicing in the finite PA-matching case.

A *splicing rule* over V is a quadruple $r = (u_1, u_2, u_3, u_4)$, with $u_i \in V^*, 1 \leq i \leq 4$.

Given a finite set R of splicing rules and the strings $x, y \in V^*$ we write

$$\begin{aligned} \sigma_R(x,y) \;\; = \;\; & \{x_1u_1u_4y_2 \mid x = x_1u_1u_2x_2,\ y = y_1u_3u_4y_2, \\ & (u_1,u_2,u_3,u_4) \in R,\ x_1,x_2,y_1,y_2 \in V^*\}. \end{aligned}$$

For $L_1, L_2, L \subseteq V^*$, we define

$$\begin{aligned} &\sigma_R(L_1, L_2) = \bigcup_{x \in L_1, y \in L_2} \sigma_R(x,y) \\ &\sigma_R(L) = \sigma_R(L, L), \\ &\sigma_R^0(L) = L, \\ &\sigma_R^{i+1}(L) = \sigma_R^i(L) \cup \sigma_R(\sigma_R^i(L)),\ i \geq 0, \\ &\sigma_R^*(L) = \bigcup_{i \geq 0} \sigma_R^i(L). \end{aligned}$$

Note that in the splicing case we cannot check the suffix-prefix matching; this is the main difference between the two operations. However, with the use of other operations, the two operations can simulate each other.

Lemma 4.1.1 *If a family F of languages is closed under concatenation with symbols and non-iterated splicing, then F is closed under the operation $fPAm$.*

Proof. For $L_1, L_2 \subseteq V^*$, consider two symbols c_1, c_2 not in V. For a finite set $X \subseteq V^+$, consider the set of splicing rules $R = \{(\varepsilon, xc_2, c_1x, \varepsilon) \mid x \in X\}$. Then we obviously have

$$PAm_X(L_1, L_2) = \sigma_R(L_1\{c_2\}, \{c_1\}L_2),$$

which implies the lemma. □

Lemma 4.1.2 *If a family F of languages is closed under finite transductions and the operation fPAm, then it is closed under non-iterated splicing.*

Proof. Let $L_1, L_2 \subseteq V^*$ be two languages and R be a finite set of splicing rules over V. For each rule $r = (u_1, u_2, u_3, u_4)$ consider a new symbol a_r and let $X = \{a_r \mid r \in R\}$ be the set of these symbols.

We define two finite transducers, M_1, M_2, such that, for each $x \in V^*$,

$$\begin{aligned} T_{M_1}(x) &= \{x_1 u_1 a_r \mid x = x_1 u_1 u_2 x_2, \text{ for } r = (u_1, u_2, u_3, u_4) \in R \\ &\qquad \text{and } x_1, x_2 \in V^*\}, \\ T_{M_2}(x) &= \{a_r u_4 x_2 \mid x = x_1 u_3 u_4 x_2, \text{ for } r = (u_1, u_2, u_3, u_4) \in R \\ &\qquad \text{and } x_1, x_2 \in V^*\}. \end{aligned}$$

Clearly, the equality

$$\sigma_R(L_1, L_2) = PAm_X(T_{M_1}(L_1), T_{M_2}(L_2))$$

holds (the PA-matching just puts together the strings marked by the two tranducers) which proves the lemma. □

Of course, the concatenation with symbols can also be performed by finite transducers, therefore, by combining the above two lemmas we get:

Theorem 4.1.1 *If F is a family of languages closed under finite transductions, then F is closed under the operation fPAm if and only if it is closed under non-iterated splicing.*

Then by Theorem 7.1 from [104], we get the following corollary:

Corollary 4.1.1 *The families REG, CF, RE are closed under the fPAm operation, but LIN is not closed.*

Also the family CS is closed under the operation $fPAm$ (although it is not closed under splicing), as a consequence of the following result.

Lemma 4.1.3 *If a family F of languages is closed under concatenation, union, and right and left derivatives, then F is closed under the operation $fPAm$.*

Proof. The following equality is obvious:

$$PAm_X(L_1, L_2) = \bigcup_{x \in X} \partial_x^r(L_1)\partial_x^l(L_2).$$

The required closure properties of F imply then the lemma. □

Corollary 4.1.2 *The family CS is closed under the $fPAm$ operation.*

We move now to investigate the properties of arbitrary PA-match operation.

Lemma 4.1.4 *If a family F of languages is closed under the shuffle and finite transductions, then F is closed under PAm.*

Proof. Let $L_1, L_2 \in F, L_1, L_2 \subseteq V^*$. Consider the alphabet $V' = \{a' \mid a \in V\}$ and the morphism h defined by $h(a) = a'$, for $a \in V$. Since each morphism can be realized by a finite transducer, $h(L_1) \in F$.

We construct now a finite transducer M which, informally speaking, works as follows on the strings from the language $Shuf(L_1, h(L_2))$:

- M reads a prefix of the input string formed exclusively by non-primed letters and leaves it unchanged;
- then, starting from a new state, M checks for a while if the input contains only pairs of letters of the form aa', and writes nothing to the output;
- then, starting from another state, M reads only primed symbols and writes as output the non-primed versions of them.

It is easy to see that M defines a transduction that satisfies the equation

$$T_M(Shuf(L_1, h(L_2))) = PAm(L_1, L_2).$$

Thus the lemma holds. □

Lemma 4.1.5 *If a family F of languages such that $REG \subseteq F$ is closed under concatenation with symbols, left derivatives, and PAm, then F is closed under $Pref$.*

Proof. Let $L \subseteq V^*$ and let c_1, c_2 be two new symbols. Then obviously

$$Pref(L) = \partial^l_{c_1} PAm(\{c_1\}L\{c_2\}, V^*\{c_2\}),$$

and so the lemma holds. □

Theorem 4.1.2 *1. The families REG and RE are closed under PAm.*

2. The families LIN, CF, and CS are not closed under PAm.

Proof. Let us consider the languages

$$L_1 = \{c_1 w d_1 mi(w) d_2 \mid w \in \{a,b\}^+\},$$
$$L_2 = \{d_1 w d_2 mi(w) c_2 \mid w \in \{a,b\}^+\}.$$

Clearly, both of them are linear languages. It is easy to see that

$$\partial^l_{c_1}(\partial^r_{c_2}(PAm(L_1, L_2))) = \{ww \mid w \in \{a,b\}^+\},$$

which is not a context-free language. Consequently, the families LIN and CF are not closed under PAm.

The family CS is not closed under $Pref$; the families REG, RE are closed under shuffle and finite transductions. Thus, the theorem follows from the previous lemmas. □

A language L is said to be a *fixed point* of the PA-match operation iff $PAm(L) = L$.

If L is a regular language, then by Theorem 4.1.2 we have that $PAm(L)$ is regular. The equivalence problem for regular languages is decidable. Therefore, we can decide whether or not a given regular language is a fixed point of the PA-match operation. As expected, this is not true for the family of context-free languages.

Theorem 4.1.3 *The problem whether or not a given context-free language is a fixed point of the PA-match operation is undecidable.*

Proof. Take two arbitrary n-tuples of nonempty strings over the alphabet $\{a,b\}$, $x=(x_1,x_2,\ldots,x_n), y=(y_1,y_2,\ldots,y_n)$, $n\geq 1$, and consider the languages

$$\begin{aligned}
L_z &= \{ba^{t_1}ba^{t_2}\ldots ba^{t_k}cz_{t_k}\ldots z_{t_2}z_{t_1} \mid k\geq 1, 1\leq t_i\leq n,\\
&\quad 1\leq i\leq k\}, \text{ for } z\in\{x,y\},\\
L_s &= \{w_1cw_2cmi(w_2)cmi(w_1)|w_1,w_2\in\{a,b\}^*\},\\
L(x,y) &= \{a,b,c\}^* - (L_x\{c\}mi(L_y)\cap L_s).
\end{aligned}$$

It is known, see, e.g., [117], that $L(x,y)$ is a context-free language.

If $PCP(x,y)$ has no solution, then $L(x,y)=\{a,b,c\}^*$ and

$$PAm(L(x,y))=\{a,b,c\}^*. \tag{4.1}$$

If $PCP(x,y)$ has solutions, then $L(x,y)\neq\{a,b,c\}^*$ but equality 4.1 still holds. (For each $w\in\{a,b,c\}^*$, the strings c^4 and c^4w are in $\{a,b,c\}^*$ but not in L_s; hence, these strings are in $L(x,y)$. This means that $w\in PAm(c^4,c^4w)$, that is, $\{a,b,c\}^*\subseteq PAm(L(x,y))$. The converse inclusion is trivial.)

Consequently, $PAm(L(x,y))=L(x,y)$ if and only if $PCP(x,y)$ has no solution. Since PCP is undecidable, the theorem holds. □

4.1.2 The Iterated Case

We will investigate now the iterated version of the PA-match operation.

It is defined as follows. For a language $L\subseteq V^*$ and a finite set $X\subseteq V^+$, we define:

$$\begin{aligned}
PAm_X^0(L) &= L,\\
PAm_X^{k+1}(L) &= PAm_X^k(L)\cup PAm_X(PAm_X^k(L)),\ k\geq 0,\\
PAm_X^*(L) &= \bigcup_{k\geq 0} PAm_X^k(L).
\end{aligned}$$

When X is finite, the iterated PA-mathching operation is denoted by $fPAm^*$; in the case $X=V^*$, the corresponding operation is denoted by PAm^*.

Lemma 4.1.6 *If a family F of languages is closed under concatenation with symbols, iterated splicing, and left and right derivatives, then F is closed under iterated finite PA-matching.*

Proof. Let $L \subseteq V^*$ be a language in F and X be a finite subset of V^+. Let c_1, c_2 be two new symbols. We associate with X the set of splicing rules $R = \{(\varepsilon, xc_2, c_1x, \varepsilon) \mid x \in X\}$. Clearly,

$$PAm_X^*(L) = \partial_{c_1}^l(\partial_{c_2}^r(\sigma_R^*(\{c_1\}L\{c_2\}))).$$

Hence the lemma holds. □

Lemma 4.1.7 *Let F be a family of languages closed under concatenation with symbols, union, left and right derivatives.*

1. *If F is closed under $fPAm^*$, then F is closed under $fPAm$.*
2. *If F is closed under PAm^*, then F is closed under PAm.*

Proof. For $L_1, L_2 \subseteq V^*$, let c_1, c_2 be two new symbols. It is easy to see that the following equation holds:

$$PAm_X(L_1, L_2) = \partial_{c_1}^l(\partial_{c_2}^r(PAm_X^*(\{c_1\}L_1 \cup L_2\{c_2\}))).$$

(The derivatives require that at least one PAm operation is performed, while the markers c_1, c_2 prevent performing more than one such operation.) Note that this relation holds also for PAm. □

Theorem 4.1.4 1. *The families REG and RE are closed under both $fPAm^*$ and PAm^*.*

2. *The family LIN is not closed under $fPAm^*$ and PAm^*.*

3. *The family CF is closed under $fPAm^*$ but it is not closed under PAm^*.*

4. *The family CS is closed neither under $fPAm^*$ nor under PAm^*.*

Proof. 1. The closure under $fPAm^*$ follows from Lemma 4.1.6 and the fact that the family of regular languages is closed under iterated splicing (see [26, 60, 104]).

A more involved argument is required for proving the closure under PAm^* (remember that the regularity is not preserved by an iterated splicing with respect to a regular set of splicing rules – see [102]).

Let $R \subseteq V^*$ be a regular language recognized by a finite automaton $M = (Q, V, \delta, q_0, F)$, which satisfies the following conditions:

$$\begin{aligned} F &= \{q_f\},\ q_0 \neq q_f, \\ \delta(q_f, a) &= \emptyset, \text{ for all } a \in V, \\ q_0 &\notin \delta(q_0, x), \text{ for each } x \in V^+. \end{aligned}$$

Clearly, each regular language is accepted by a finite automaton satisfying the above conditions.

We construct now iteratively a sequence of finite automata with ε-moves, $M_0, M_1, \ldots, M_i, \ldots$ with $M_i = (Q, V, \delta_i, q_0, \{q_f\})$ as follows:

- $M_0 = (Q, V, \delta_0, q_0, \{q_f\}) = M$.
- $M_{i+1} = (Q, V, \delta_{i+1} q_0, \{q_f\})$ is obtained from M_i as follows.
  - $\delta_{i+1}(s, a) = \delta_i(s, a)$, for all $s \in Q, a \in V \cup \{\varepsilon\}$.
  - For all pairs of different states $q, q' \in Q - \{q_0, q_f\}$ such that:
    1. $q \notin \delta_i(q', \varepsilon)$,
    2. $L(M_q) \cap L(M^{q'}) \neq \emptyset$,

    where

    $$M_q = (Q, V, \delta_i, q_0, \{q\}), \text{ and } M^{q'} = (Q, V, \delta_i, q', \{q_f\}),$$

    we set

    $$\delta_{i+1}(q', \varepsilon) = \delta_{i+1}(q', \varepsilon) \cup \{q\}.$$

Obviously, the above sequence is finite, because there exists k such that $M_{k+1} = M_k$ (the set of states is not changed, only new transitions are added); hence $M_{k+p} = M_k$, for all $p \geq 0$. Note also that the construction is effective due to the decidability of the emptiness problem for the intersection of two regular languages. Furthermore,

$$R = Acc(M_0) \subseteq Acc(M_1) \subseteq Acc(M_2) \subseteq \ldots \subseteq Acc(M_k) =$$

$$Acc(M_{k+1}) = \ldots \subseteq PAm^*(R)$$

holds. On the other hand, one may easily prove by induction that $PAm^j(R) \subseteq Acc(M_j)$, for all $j \geq 0$; therefore $PAm^*(R) = Acc(M_k)$.

2. Because the family LIN is closed neither under $fPAm$ (Corollary 4.1.1) nor under PAm (Theorem 4.1.2), by Lemma 4.1.7 it follows that it is not closed under the iterated versions of these operations.

3. It is known that the family CF is closed under iterated splicing [60]; thus, the closure of CF under $fPAm^*$ follows from Lemma 4.1.6. By Lemma 4.1.7 and Theorem 4.1.2, we get the non-closure of CF under PAm^*.

4. Consider now a language $L \in RE - CS, L \subseteq V^*$. There are $a_1, a_2 \notin V$ and a context-sensitive language $L' \subseteq L\{a_1\}\{a_2\}^*$ such that for each $w \in L$ there is $i \geq 0$ with $wa_1a_2^i \in L'$. We have then

$$\partial^r_{a_1}(PAm^*_{\{a_2,a_2c\}}(L'\{a_2, a_2^2\} \cup \{a_2c\}) = L.$$

Indeed, the first PAm operation transforms strings $wa_1a_2^n \in L'$ into $wa_1a_2^{n-1}c$. The next step leads to $wa_1a_2^{n-2}$ and the process can be iterated. The right derivative with respect to a_1 selects from $PAm^*_{\{a_2,a_2c\}}(L'\{a_2, a_2c\} \cup \{a_2c\})$ the strings of the form wa_1. Since we nondeterministically concatenate L' with both a_2 and a_2^2, in this way we can get wa_1 for all $w \in L$. Thus, the equality follows.

If the family CS was closed under the operation $PAm^*_{\{a_2,a_2c\}}$, then $L \in CS$, which is a contradiction. □

As a matter of fact, the non-closure of the families CF and CS under iterated arbitrary PA-matching may be obtained from a more general result.

Theorem 4.1.5 *Each recursively enumerable language $L \subseteq V^*$ can be written as $L = \partial^l_{c_1}(\partial^r_{c_2}(PAm^*(L') \cap \{c_1\}V^*\{c_2\}))$, where L' is a context-free language and c_1, c_2 are two new symbols.*

Proof. Assume that L is generated by a type-0 grammar $G = (N, V, S, P)$ in the Geffert normal form, that is, with $N = \{S, A, B, C\}$ and P having only context-free rules of the form $S \rightarrow x, x \in (N \cup V)^+$, and a single extra rule $ABC \rightarrow \varepsilon$. Consider the context-free grammar $G' = (\{S\}, V \cup \{A, B, C, X\}, S, \{S \rightarrow h(x) \mid S \rightarrow x \in P\})$, where X

is a new symbol, and h is a morphism that replaces A by XA leaving all the other symbols unchanged. Consider the language

$$\begin{aligned} L' &= \{c_1\}L(G')\{c_2\} \cup \{XABCwc_2Ymi(w)Z \mid w \in (V \cup \{B,C\})^*\} \\ &\cup \{YwZmi(w)c_2 \mid w \in (V \cup \{B,C\})^*\}, \end{aligned}$$

where Y, Z are two new symbols. Clearly, L' is a context-free language.

Let $c_1w_1XABCw_2c_2$ be a string in $\{c_1\}L(G')\{c_2\}$, with $w_2 \in (V \cup \{B,C\})^*$ (that is, this is the rightmost occurrence of $XABC$ in our string). The only possible PA-matching operation is

$$PAm(c_1w_1XABCw_2c_2, XABCw_2c_2Ymi(w_2)Z) = c_1w_1Ymi(w_2)Z.$$

The obtained string can again "enter" only one operation:

$$PAm(c_1w_1Ymi(w_2)Z, Ymi(w_2)Zmi(mi(w_2))c_2) = c_1w_1w_2c_2.$$

In this way, one occurrence of $XABC$ has been removed. By iterating the PAm operation, all such substrings can be removed – therefore $\{c_1\}L\{c_2\} = PAm^*(L') \cap \{c_1\}V^*\{c_2\}$ holds. The left and the right derivatives lead now to L. □

As a direct consequence of the above result, we find that every family of languages that contains all context-free languages but not all recursively enumerable languages, and is closed under intersection with regular sets and right and left derivatives, is not closed under PAm^*. This is the case for the most of the language families in the regulated rewriting area [114]. Moreover, the above result implies some undecidability results.

Corollary 4.1.3 *The following problems are undecidable:*

1. *For an arbitrary $L \in CF$, is $PAm^*(L)$ regular/context-free?*

3. *For an arbitrary $L \subseteq V^*, L \in CF$, does $w \in V^*$ belong to $PAm^*(L)$?*

4.2 The Overlapping Operation

In this section we consider another operation on languages that may be viewed as the dual of PA-matching. While the PA-matching operation removes the matched part, the *overlapping* operation preserves the matched part and removes the rest.

More precisely, for strings x, y we define

$$Ov(x,y) = PSuf(x) \cap PPref(y).$$

Then,

$$Ov(L_1, L_2) = \bigcup_{x \in L_1, y \in L_2} Ov(x,y) = PSuf(L_1) \cap PPref(L_2).$$

We write $Ov(L)$ instead of $Ov(L, L)$. The closure properties of the language families in the Chomsky hierarchy under the overlapping operation are the same as for the PA-matching operation.

Theorem 4.2.1 1. *The families REG and RE are closed under Ov.*
2. *The families LIN, CF, and CS are not closed under Ov.*

Proof. The first assertion follows from the closure of both families under intersection, $PPref$ and $PSuf$.

It is easy to see that the closure under overlapping, together with other "easy" closure properties (concatenation with symbols, left and right derivatives), implies the closure under intersection ($L_1 \cap L_2 = \partial^l_{c_1}(\partial^r_{c_2}(Ov(\{c_1^2\}L_1\{c_2\}, \{c_1\}L_2\{c_2^2\})))$ and the prefix operation ($Pref(L) = \partial^l_c(\{c^2\}V^*, \{c\}L\{c\})$). These observations imply the second claim. □

From the previous proof it follows that the fixed point problem for Ov is decidable for regular languages. The problem remains undecidable for context-free languages (with the same proof as for PAm).

Now, let us consider the iterated version of the overlapping operation. The usual way of defining an iterated operation (see the case of the splicing and the case of PA-matching) does not work for the iterated overlapping, because $Ov(Ov(L)) \subseteq Ov(L)$, which makes the usual definition ($Ov^{k+1}(L) = Ov^k(L) \cup Ov(Ov^k(L))$) uninteresting. Therefore, we shall define $Ov^{k+1}(L) = Ov(Ov^k(L))$, for all $k \geq 1$. Moreover, $Ov^*(L) = L'$ iff the following two conditions are fulfilled:

(i) $L' \subseteq Ov^k(L)$, for all $k \geq 1$,

(ii) for each L'' with $L' \subset L''$ there exists $k \geq 1$ such that $L'' \not\subseteq Ov^k(L)$.

This means that, $Ov^*(L)$ is the largest language (with respect to inclusion) which is included in all the sets $Ov(L), Ov^2(L), \ldots$

Theorem 4.2.2 1. *For each $k \geq 1$ there is a language L_k such that $Ov^*(L_k) = Ov^k(L_k)$.*

2. *There are languages L such that $Ov^{k+1}(L) \subset Ov^k(L)$, for all $k \geq 1$.*

Proof. Consider the language $L_k = \{a^ib^j \mid 1 \leq i, j \leq k\}$. It is easy to see that $Ov(L_m) = L_{m-1}$, $2 \leq m \leq k$, and $Ov(L_1) = \emptyset$. Therefore, $Ov^*(L_k) = Ov^k(L_k)$.

Consider also the language

$$L_\infty = \bigcup_{n \geq 1} \{(ba^nb)^i(ca^nc)^j \mid 1 \leq i, j \leq n\}.$$

For each n, we can overlap only strings containing blocks ba^nb, ca^nc. For given n, we can perform a bounded number of overlappings, because at each step we have to remove either the prefix ba^nb or the suffix ca^nc. Therefore $Ov^{k+1}(L_\infty) \neq Ov^k(L_\infty)$ for $k \leq n$. Because n can be arbitrarily large, the operation can be iterated an arbitrarily large number of steps. □

Note that $Ov^k(L_\infty) \neq \emptyset$, but $Ov^*(L_\infty) = \emptyset$.

Theorem 4.2.3 *The families LIN and CF are not closed under Ov^*.*

Proof. For $L_1, L_2 \subseteq V^*$, let us consider two new symbols, c_1, c_2. We obtain the equality:

$$Ov^*(\{c_1\}^*L_1\{c_2\}^* \cup \{c_1\}^*L_2\{c_2\}^*) \cap \{c_1\}V^*\{c_2\} = \{c_1\}(L_1 \cap L_2)\{c_2\}.$$

Indeed, $\{c_1\}^*(L_1 \cap L_2)\{c_2\}^* \subseteq Ov(\{c_1\}^*L_1\{c_2\}^* \cup \{c_1\}^*L_2\{c_2\}^*)$. Starting from strings in $\{c_1\}^*(L_1 \cap L_2)\{c_2\}^*$, we can iterate the overlapping operation an arbitrarily large number of times.

By this equation, the closure under Ov^* implies the closure under intersection. Since the families LIN and CF are not closed under intersection (but they are closed under concatenation with regular languages, intersection with regular languages, union, and left and right derivatives), the theorem holds. □

Theorem 4.2.4 *The family CS is not closed under Ov^*.*

Proof. Let $L \subseteq V^*$ and let c_1, c_2 be new symbols (not in V). Consider the language

$$L' = \{c_1\}^*V^*\{c_2\}^* \cup \{c_1\}^*((Shuf(L, \{c_2\}^*)) \cap V^*\{c_2\}^*V^*).$$

This is a context-sensitive language. It is easy to see that

$$Ov^*(L') \cap \{c_1\}V^*\{c_2\} = \{c_1\}Pref(L)\{c\}.$$

(We have $\{c_1\}^*Pref(L)\{c_2\}^* \subseteq Ov(L')$, hence we can iterate the operation Ov an arbitrarily large number of times.)

Because the family CS is closed under right and left derivatives, but not under the operation $Pref$, we obtain the non-closure under Ov^*. □

Clearly, RE is closed under the iterated overlapping operation. The case of the family REG will be settled below (also in affirmative), after establishing two auxiliary results.

Let $A = (Q, V, \delta, q_0, F)$ be a minimal complete deterministic finite automaton; because the automaton is complete, the mapping δ is total and a dead state exists from which there is no path to a final state. Let $\mathcal{A}$ be the set of all finite automata of the form $A_{p,q} = (Q, V, \delta, p, \{q\})$, for $p, q \in Q$. Clearly, this is a finite set. We denote by $L(\mathcal{A})$ the family of all languages recognized by automata in $\mathcal{A}$ and by $CL(\mathcal{A})$ the closure of the family $L(\mathcal{A})$ under finite union and finite intersection operations. Because $L(\mathcal{A})$ is a finite family, also $CL(\mathcal{A})$ is a finite family of languages.

Lemma 4.2.1 *The family $CL(\mathcal{A})$ is closed under complementation.*

Proof. Let L be a language in $CL(\mathcal{A})$. It can be written in the form $L = (L_{1,1} \cap \ldots \cap L_{1,n_1}) \cup \ldots \cup (L_{m,1} \cap \ldots \cap L_{m,n_m})$, where each language $L_{i,j}, 1 \leq i \leq m, 1 \leq j \leq n_i$, is an element of $L(\mathcal{A})$. The complement of each language $L_{i,j}$ (we denote the complement of a language K by $\overline{K}$) is also in $L(\mathcal{A})$, since A was a complete deterministic automaton. Because $\overline{L} = (\overline{L_{1,1}} \cup \ldots \cup \overline{L_{1,n_1}}) \cap \ldots \cap (\overline{L_{m,1}} \cup \ldots \cup \overline{L_{m,n_m}})$, it follows that also the complement of L is in $CL(\mathcal{A})$. □

Lemma 4.2.2 *The family $CL(\mathcal{A})$ is closed under the non-iterated overlapping operation.*

Proof. Let L be a language in $CL(\mathcal{A})$. We write it in the form $L = T_1 \cup \ldots \cup T_n$, where each $T_i, 1 \leq i \leq n$ is a finite intersection of languages in $CL(\mathcal{A})$. For every integer $i = 1, \ldots, n$, denote:

$$\begin{aligned} K_i &= \{x \in V^* \mid \partial^l_x(T_i) = \{\varepsilon\}\}, \\ M_i &= \{x \in V^* \mid \partial^l_x(T_i) = \emptyset\}, \\ P_i &= \{x \in V^* \mid \partial^r_x(T_i) = \{\varepsilon\}\}, \\ R_i &= \{x \in V^* \mid \partial^r_x(T_i) = \emptyset\}. \end{aligned}$$

Note that the following assertions hold for each string $x \in V^*$:

$$\begin{aligned} & x \in L - Ov(L) \Leftrightarrow \partial^l_x(L) = \{\varepsilon\} \text{ or } \partial^r_x(L) = \{\varepsilon\} \Leftrightarrow \\ & (\partial^l_x(T_i) \subseteq \{\varepsilon\} \text{ for all } i \text{ and there is some } j \text{ such that } \partial^l_x(T_j) \neq \emptyset), \\ & \qquad \text{or} \\ & (\partial^r_x(T_i) \subseteq \{\varepsilon\} \text{ for all } i \text{ and there is some } j \text{ such that } \partial^r_x(T_j) \neq \emptyset). \end{aligned}$$

Consequently, we have

$$\begin{aligned} L - Ov(L) \;=\; & \left(\bigcap_{1 \leq i \leq n} (K_i \cup M_i) - \bigcap_{1 \leq i \leq n} M_i \right) \\ \cup \; & \left((\bigcap_{1 \leq i \leq n} (P_i \cup R_i) - \bigcap_{1 \leq i \leq n} R_i \right). \end{aligned}$$

By Lemma 4.2.1, it suffices to prove that for every $i = 1, \ldots, n$ the languages K_i, M_i, P_i, R_i are contained in $CL(\mathcal{A})$.

Consider any $T_i = L_1 \cap \ldots \cap L_m$, where $L_j \in L(\mathcal{A}), 1 \leq j \leq m$. For every j there is an automaton $A_j = (Q, V, \delta, p_j, \{f_j\})$ in $\mathcal{A}$ such that $L_j = L(A_j)$.

a. In the standard manner, construct the product automaton $A_K = (Q^m, V, \vec{\delta}, \vec{p}, \{\vec{f}\})$ which accepts T_i, where $\vec{p} = (p_1, \ldots, p_m)$ and $\vec{f} = (f_1, \ldots, f_m)$. If A_K has a cycle which contains the final state

$\vec{f}$, then $K_i = \emptyset$ and this is a language in $CL(\mathcal{A})$. Otherwise, $K_i = T_i$, which is again in $CL(\mathcal{A})$.

b. For the previous automaton A_K, let F_M be the set of states which appear on a path from $\vec{p}$ to $\vec{f}$. Consider the automaton $A_M = (Q^m, V, \vec{\delta}, \vec{p}, F_M)$. Then,

$$\overline{M_i} = Acc(A_M) = \bigcup_{\vec{q} \in F_M} Acc((Q^m, V, \vec{\delta}, \vec{p}, \{\vec{q}\})).$$

For every state $\vec{q} = (q_1, \ldots, q_m) \in F_M$, the language accepted by the automaton $(Q^m, V, \vec{\delta}, \vec{p}, \{\vec{q}\})$ is an intersection of the languages $Acc((Q, V, \delta, p_j, \{q_j\}))$, which are in $L(\mathcal{A})$ for all $j = 1, \ldots, m$. Consequently, $M_i \in CL(\mathcal{A})$.

c. The proof of the relation $R_i \in CL(\mathcal{A})$ can be obtained in the same manner as in the case of M_i.

d. For each automaton $A_j = (Q, V, \delta, p_j, \{f_j\})$ as above we construct its reversal, $A_j^R = (Q, V, \delta^R, f_j, \{p_j\})$ and we make A_j^R deterministic by the usual subset construction technique. Let $A_{j*}^R = (2^Q, V, \delta_*^R, \{f_j\}, F_j)$ be the automaton obtained in this way. Then, by definition, $Acc(A_{j*}^R) = mi(L_j)$ holds.

We construct the product $((2^Q)^m, V, \vec{\delta_*^R}, \vec{s}, F)$ of automata A_{1*}^R, …, A_{m*}^R, which accepts $T_i^R = mi(L_1) \cap \ldots \cap mi(L_m)$, where $\vec{s} = (\{f_1\}, \ldots, \{f_m\})$. Let F_P be the set of all states $\vec{q}$ in F such that $\vec{q}$ does not have a path to any state of F. Let $A_P = ((2^Q)^m, V, \vec{\delta_*^R}, \vec{s}, F_P)$. Then, we have $P_i^R = L(A_P)$. Thus,

$$\begin{aligned} P_i &= Acc(A_P^R) \\ &= Acc(((2^Q)^m, V, (\vec{\delta_*^R})^R, F_P, \{\vec{s}\})) \\ &= \bigcup_{\vec{q} \in F_P} Acc(((2^Q)^m, V, (\vec{\delta_*^R})^R, \vec{q}, \{\vec{s}\})). \end{aligned}$$

It suffices now to show that for every $\vec{q} \in F_P$, the language $Z_{\vec{q}} = Acc(((2^Q)^m, V, (\vec{\delta_*^R})^R, \vec{q}, \{\vec{s}\}))$ is in $CL(\mathcal{A})$. Let $\vec{q} = (E_1, \ldots, E_m) \in$

F_P, where $E_j \subseteq Q, 1 \leq j \leq m$. Note that for any $(X_1, \ldots, X_m)$ and $(Y_1, \ldots, Y_m)$ in $(2^Q)^m$ and for any $a \in V$, the following assertions hold:

$$\begin{aligned}
& (Y_1, \ldots, Y_m) \in (\overrightarrow{\delta_*^R})^R((X_1, \ldots, X_m), a) \\
\Leftrightarrow\ & \overrightarrow{\delta_*^R}((Y_1, \ldots, Y_m), a) = (X_1, \ldots, X_m) \\
\Leftrightarrow\ & \delta_*^R(Y_j, a) = X_j, \text{ for all } j = 1, \ldots, m, \\
\Leftrightarrow\ & X_j = \{q \mid q \in \delta^R(p, a), \text{ for some } p \in Y_j\}, \text{ for all } 1 \leq j \leq m, \\
\Leftrightarrow\ & X_j = \{q \mid p = \delta(q, a), \text{ for some } p \in Y_j\}, \text{ for all } 1 \leq j \leq m.
\end{aligned}$$

Therefore, we have

$$Z_{\vec{q}} = \bigcap_{1 \leq j \leq m} \left(\bigcap_{r \in E_j} Acc((Q, V, \delta, r, \{f_j\})) - \bigcup_{r \notin E_j} Acc((Q, V, \delta, r, \{f_j\})) \right)$$

hence this language is in $CL(\mathcal{A})$ and this completes the proof of the lemma. □

Theorem 4.2.5 *The family REG is closed under the operation Ov^*.*

Proof. Starting from a minimal deterministic finite automaton A for a regular language L, we construct the family $CL(\mathcal{A})$ as above. Because this family is closed under non-iterated overlapping, all languages $Ov^k(L), k \geq 1$, are in $CL(\mathcal{A})$. Because the family $CL(\mathcal{A})$ is finite, it follows that only finitely many languages $Ov^k(L)$ are different to each other. The smallest of them is equal to $Ov^*(L)$ and it is an element of $CL(\mathcal{A})$. It follows that $Ov^*(L)$ is a regular language. □

4.3 Operations Suggested by Gene Assembly in Ciliates

We define three operations on strings and languages suggested by the process of gene assembly in ciliates. The closure of the classes of regular and context-free languages under these operations is settled. We also consider the macronuclear language of a given language. Finally, some open problems and further directions of research are discussed.

A bit more precisely, these operations are suggested by the intricate transformation process by which the macronucleus of a ciliate results from its micronucleus. The reader interestead in more biological details is refered to [76] and [106].

Following [42, 43] we define the following three operations which might be viewed as formal linguistical definitions of the operations through the gene assembly process in cilitates is accomplished:

- *(loop, direct repeat)-excision* (*ld*, for short),
- *(hairpin, inverted repeat)-excision* (*hi*, for short),
- *(double loop, alternating direct repeat)-excision* (*dlad*, for short).

The computational power of this transformational process taking place in ciliates has been considered in [77] and [42].

A *gac*-scheme (*gene assembly in ciliates)* is a pair $\sigma = (V, P)$, where V is an alphabet and P is a finite subset of V^+, whose elements are called *pointers*, such that $mi(P) = P$.

For a *gac*-scheme as above, we write $\bar{V} = \{\bar{a} \mid a \in V\}$ and $\bar{P}$ for the set consisting of all strings obtained from those of P by replacing each letter with its barred copy in $\bar{V}$.

4.3.1 The *ld* Operation

The Non-Iterated Case

Given a *gac*-scheme $\sigma = (V, P)$ and a string $w \in V^+$, we define formally the *ld* operation as follows.

The *ld* operation proceeds informally as shown in Figure 4.1. As one can see, the linear molecule (string), in which two occurrences of a pointer α have been emphasized, is folded into a loop aligned by this pair of pointers. Then, the string is cut as in Figure 4.1, yielding two strings, a linear one and a circular one. We consider here that a circular molecule is no longer a micronuclear precursor for another assembly, so that we keep only the linear molecule. For this reason, the segment y must not contain any pointer.

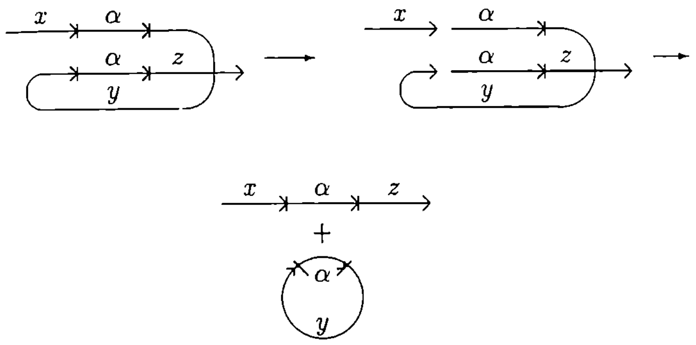

Figure 4.1.

Formally, the *ld* operation is defined by

$$\begin{aligned} ld_\sigma(w) &= \{x\alpha z \mid w = x\alpha y\alpha z,\ x,z \in V^*, y \in V^+, \\ &\quad \alpha \in P,\ Sub(y) \cap P = \emptyset\}. \end{aligned}$$

A string w is called an ld_σ-*macronuclear string* if $ld_\sigma(w) = \emptyset$.

The above operation can be extended to languages in a natural way:

$$ld_\sigma(L) \cup_{w \in L} ld_\sigma(w).$$

A family of languages $\mathcal{F}$ is closed under the operation *ld* if for any language $L \in \mathcal{F}$ and any *gac*-scheme σ, $ld_\sigma(L) \in \mathcal{F}$ holds.

Proposition 4.3.1. *Any full trio is closed under ld.*

Proof. Let $\sigma = (V, P)$ be a *gac*-scheme with $P = \{x_1, x_2, \ldots, x_n\}$ for some n. The construction is rather simple: We define the homomorphisms

$$h : (V \cup \bar{V} \cup \{c_1, c_2, \ldots, c_n, d_1, d_2, \ldots, d_n\})^* \longrightarrow V^*,$$

defined by

$$\begin{aligned} h(a) &= h(\bar{a}) = a,\ a \in V, \\ h(c_i) &= h(d_i) = x_i,\ 1 \le i \le n. \end{aligned}$$

and

$$g : (V \cup \bar{V} \cup \{c_1, c_2, \ldots, c_n, d_1, d_2, \ldots, d_n\})^* \longrightarrow V^*,$$

defined by

$$\begin{aligned} g(a) &= a,\ g(\bar{a}) = l,\ a \in V, \\ g(c_i) &= x_i,\ g(d_i) = l,\ 1 \leq i \leq n. \end{aligned}$$

Now we consider the regular language

$$R = \cup_{i=1}^{n} V^*\{c_i\}(\bar{V}^+ \setminus \bar{V}^*\bar{P}\bar{V}^*)\{d_i\}V^*.$$

We claim that

$$ld_\sigma(L) = g(h^{-1}(L) \cap R),$$

for any language L over V. Indeed, the regular language R assures that the following conditions are satisfied:

- The strings in $h^{-1}(L) \cap R$ are produced from those strings in L having two occurrences of some pointer, say x_i, whose inverse homomorphical images are the symbols c_i and d_i while the other pointer occurrences are not transformed into symbols in the set $\{c_1, c_2, \ldots, c_n, d_1, d_2, \ldots, d_n\}$.

- The segment between these two occurrences in the original string contains no occurrence of any pointer and is transformed into its barred version by the inverse homomorphism h^{-1}.

- The prefix and suffix of the original string before and after these occurrences, respectively, are left unchanged by applying h^{-1}.

Now the homomorphism g erases any symbol d_i, $1 \leq i \leq n$, together with all barred letters, restores any string x_i for c_i, $1 \leq i \leq n$, and leaves unchanged the letters from V, which concludes the reasoning. □

Since the families of regular and context-free languages are full trios, we get:

Corollary 4.3.1. *The families of regular and context-free languages are closed under the operation ld.*

The Iterated Case

Let L be a language over an alphabet V and $\sigma = (V, P)$ be a *gac*-scheme. The *ld_σ-macronuclear language* of L, denoted by $ld_\sigma M(L)$, consists of all ld_σ-macronuclear strings obtained from the strings of L by applying iteratively the operation ld with respect to σ.

More formally, we define the languages

$$\begin{aligned} R_0 &= L, \\ L_0 &= \{w \in R_0 \mid ld_\sigma(w) = \emptyset\}, \\ R_{i+1} &= ld_\sigma(R_i \setminus L_i), \text{ and} \\ L_{i+1} &= \{w \in R_{i+1} \mid ld_\sigma(w) = \emptyset\},\ i \geq 0. \end{aligned}$$

Then,

$$ld_\sigma M(L) = \cup_{i\geq 0} L_i.$$

We now address the following problem: given a regular/context-free language L and a *gac*-scheme σ, is $ld_\sigma M(L)$ still regular/context-free. In the remaining part of this section we provide a complete answer for a large class of *gac*-schemata.

Lemma 4.3.1. *Let $\sigma = (V, P)$ be a gac-scheme such that any two strings $x, y \in P$, with $y \neq mi(x)$, do not overlap each other. Then, one can construct a deterministic finite transducer M such that for any language L over V, $g_M(L)$ is exactly the set of all ld_σ-macronuclear strings existing in L.*

Proof. We assume that $P = \{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}$ for some $n \geq 1$ and $x^{(j)} = x_1^{(j)} x_2^{(j)} \ldots x_{|x^{(j)}|}^{(j)}$. Before defining the finite transducer, we briefly recall the well-known KMP algorithm for string matching proposed in [69]. For each $1 \leq j \leq n$, one constructs the array $next_j$ of dimension $|x^{(j)}|$ provided by the following algorithm:

```
begin
  $next_j(1) := -1$;
  for $i := 2$ to $|x^{(j)}|$ do
    $k := next_j(i-1) + 1$;
    while $x_{i-1}^{(j)} \neq x_k^{(j)}$ and $k > 0$ do
      $k := next_j(k) + 1$;
    $next_j(i) := k$;
end
```

Given a string $y = y_1 y_2 \ldots y_m$, where $y_1, y_2, \ldots, y_m$ are symbols, the matching process of $x^{(j)}$ in y proceeds as follows. The symbols in the two strings are compared until a mismatch is found. At that point, say at $x_i^{(j)}$, the same symbol in y is compared against $x^{(j)}_{next_j(i)+1}$. If this is a mismatch too, then the same symbol in y is compared against $x^{(j)}_{next_j(next_j(i)+1)+1}$ and so forth. A special case is when the mismatch is against $x_1^{(j)}$; in this case we proceed to the next symbol in y.

Based on this brief recall of the KMP algorithm, for each $1 \le j \le n$, we define the function

$$KMP_j : \{1, 2, \ldots, |x^{(j)}|\} \times V \longrightarrow \{1, 2, \ldots, |x^{(j)}| + 1\}$$

where

$$KMP_j(i, a) = \begin{cases} i + 1, \text{ if } x_i^{(j)} = a \\ KMP_j(next_j(i) + 1), \text{ if } x_i^{(j)} \ne a \text{ and } next_j(i) \ge 0, \\ 1, \text{ otherwise} \end{cases}$$

Construct the finite transducer $M = (Q, V, \delta, \{q_0\}, F$, where

$$\begin{aligned} Q &= \{q_e\} \cup \{[\langle i_1, i_2, \ldots, i_n\rangle\langle k_1, k_2, \ldots, k_n\rangle] \mid 0 \le i_j, k_j \le |x^{(j)}| + 1, \\ &\quad 1 \le\le n\}, \\ q_0 &= [\langle 1, 1, \ldots, 1\rangle\langle 0, 0, \ldots, 0\rangle], \\ F &= Q \setminus \{q_e, q_0\} \end{aligned}$$

and the transition mapping δ is defined as follows.
For each $a \in V$, $\delta([\langle i_1, i_2, \ldots, i_n\rangle\langle k_1, k_2, \ldots, k_n\rangle], a)$ is

1. q_e, if exists $1 \le j \le n$ such that $i_j = k_j = |x^{(j)}| + 1$,

2. $[\langle i'_1, i'_2, \ldots, i'_n\rangle\langle k'_1, k'_2, \ldots, k'_n\rangle]$, otherwise, where

   - if there exists a j with $1 \le j \le n$ such that $i_j = |x^{(j)}| + 1$, then if there exists a t with $1 \le t \le n$ such that $i_t = |x^{(t)}|$

and $x^{(t)}_{i_t} = a$, then

$$\begin{aligned} i'_t &= i_t + 1, \\ i'_r &= 1, 1 \le r \ne t \le n, \\ k'_r &= 0,\ 1 \le r \le n, \end{aligned}$$

else

$$\begin{aligned} i'_j &= i_j, \\ i'_r &= KMP_r(i_r, a), 1 \le r \le n, r \ne j, \\ k'_j &= KMP_j(k_j, a), \\ k'_r &= k_r, 1 \le r \le n, r \ne j. \end{aligned}$$

- if $i_j \ne |x^{(j)}| + 1$ for all $1 \le j \le n$, then if there exists a t with $1 \le t \le n$ such that $i_t = |x^{(t)}|$ and $x^{(t)}_{i_t} = a$, then

$$\begin{aligned} i'_t &= i_t + 1, \\ i'_r &= 1, 1 \le r \ne t \le n, \\ k'_r &= 0,\ 1 \le r \le n, \end{aligned}$$

  else $i'_r = KMP_r(i_r, a)$ and $k'_r = k_r$ for all $1 \le r \le n$.

Let us give some informal explanations on the working mode of this finite transducer. As one can easily see, the transducer writes always the read letter, so that the defined transduction is a subset of the input language. Except for the error state q_e, each state is formed by a pair of n-tuples of natural numbers. When the first occurrence of a pointer, say x_i, has been meet in the input string, the corresponding number of the first component of the current state became $|x_i|+1$. By our supposition - the pointers do not overlap each other - as soon as such an occurrence has been found, the searching process for another occurrence is resumed in the first component of the current states, for all pointers other than x_i, and starts a searching process of x_i by means of the second component of the current states.

When a proximate pointer occurrence has been identified, two situations may appear:

- This is an occurrence of x_i and the transducer will enter the error state q_e, and no move is possible anymore. This corresponds to the case when the input string has two consecutive occurrences of x_i and no other pointer occurrence in between; consequently it is not an ld_σ-macronuclear string in L.

- This is an occurrence of a pointer other than x_i. In this case, the former pointer occurrence, that of x_i, is of no use for the rest of the computation, hence it can be dropped and the computation continues by considering the pointer occurrence just identified as the first pointer occurrence in the input string.

Clearly, if the transducer entirely reads the input string in a state other than the error state q_e, the input string is an ld_σ-macronuclear string in L, and we are done. □

Proposition 4.3.2. *Let $\sigma = (V, P)$ be a gac-scheme such that any two strings $x, y \in P$, with $y \neq mi(x)$, do not overlap each other. Then, for any semi-AFL $\mathcal{F}$, the language $ld_\sigma M(L)$ is in $\mathcal{F}$ provided that L is in $\mathcal{F}$.*

Proof. We assume that $P = \{x_1, x_2, \ldots, x_n\}$ for some integer $n \geq 1$. The language $T \subseteq \{1, 2, \ldots, n\}^+$ such that for every $z \in T$, $z = i_1 i_2 \ldots i_k$ the relation $i_j \neq i_{j+1}$, $1 \leq j \leq k-1$, holds, is clearly a regular language. Let $\mathcal{A} = (Q, \{1, 2, \ldots, n\}, \delta, \{q_0\}, F)$ be a deterministic finite automaton recognizing T.

We now consider the following regular languages over V

$$B_i = V^+ \setminus ((\cup_{i=1}^n V^+\{x_i\}(V^+ \setminus V^*PV^*)\{x_i\}V^+) \cup$$
$$V^*\{x_i\}(V^+ \setminus V^*PV^*)), 1 \leq i \leq n,$$

recognized by the automata $\mathcal{A}_i$, respectively,

$$E_i = V^+ \setminus ((\cup_{i=1}^n V^+\{x_i\}(V^+ \setminus V^*PV^*)\{x_i\}V^+) \cup$$
$$(V^+ \setminus V^*PV^*)\{x_i\}V^*), 1 \leq i \leq n,$$

recognized by the automata $\tilde{\mathcal{A}}_i$, respectively, and

$$I_{i,j} = B_i \cap E_j, 1 \leq i \neq j \leq n,$$

recognized by the automata $\mathcal{A}_{i,j}$, respectively.

Furthermore, we consider the regular language

$$R = \bar{V}^+ \setminus (\bar{V}^* \bar{P} \bar{V}^*),$$

the automaton recognizing R being denoted by $\mathcal{A}_R$. Assume that all the automata mentioned above have pairwise disjoint sets of states.

Claim 1. *The language*

$$\begin{aligned} L' \;=\; & \cup_{i_1 i_2 \ldots i_s \in T} B_{i_1}\{c_{i_1}\}R(\{d_{i_1}\}R)^*\{d_{i_1}\}I_{i_1,i_2}\{c_{i_2}\}R(\{d_{i_2}\}R)^* \\ & \{d_{i_2}\}I_{i_2,i_3} \ldots I_{i_{s-1},i_s}\{c_{i_s}\}R(\{d_{i_s}\}R)^*\{d_{i_s}\}E_{i_s} \end{aligned}$$

is regular.

Proof of the claim. We shall informally explain the steps of a computation of the finite automaton which recognizes L'; following these explanations, the reader can easily write a formal construction.

1. Let $j = 1$.

2. By a l-move, the automaton nondeterministically chooses a number i_j, between 1 and n, and starts to simulate the automaton $\mathcal{A}_{i_j}$ till this simulation process cannot continue. During this process of simulation, the current state is a pair consisting of the state $q_{i_j} = \delta(q_0, i_j)$ and the current state of the automaton $\mathcal{A}_{i_j}$.

   When the process stops, the current state has to be formed by q_{i_j} and a final state in $\mathcal{A}_{i_j}$, and the reading head positioned on c_{i_j}.

3. By reading c_{i_j}, the automaton starts simulating the automaton $\mathcal{A}_R$. The current state now is a pair consisting of q_{i_j} and the current state of $\mathcal{A}_R$. This process ends successfully when d_{i_j} is read in a state (q_{i_j}, r), where r is a final state in $\mathcal{A}_R$.

4. The automaton reads d_{i_j} and starts again to simulate $\mathcal{A}_R$ in the same way as in the previous step. This step will be executed till a symbol in V is read or the automaton is blocked.

5. Let $j = j + 1$.

6. Now the automaton nondeterministically chooses a symbol $i_j \neq i_{j-1}$ and may change its state into the pair (q_{i_j}, s), where $q_{i_j} = \delta(q_{i_{j-1}}, i_j)$ and s is the initial state of $\mathcal{A}_{i_{j-1}, i_j}$, or the initial state of $\bar{\mathcal{A}}_{i_j}$.

7. If the state (q_{i_j}, s) is chosen, then the automaton $\mathcal{A}_{i_{j-1}, i_j}$ is simulated as long as possible. This process must end in a state (q_{i_j}, t), with t being a final state in $\mathcal{A}_{i_{j-1}, i_j}$ and c_{i_j} as the current input symbol. Now the computation continues with step 3.

8. If the initial state of $\bar{\mathcal{A}}_{i_j}$ is chosen, then the automaton simulates the work of $\bar{\mathcal{A}}_{i_j}$ until the input string is completely read. The computation is successful (the input string is accepted) if and only if it ends in one of the final states of $\bar{\mathcal{A}}_{i_j}$.

Thus we conclude the proof of the claim.

We define the following homomorphisms:

$$h : (V \cup \bar{V} \cup \{c_1, c_2, \ldots, c_n, d_1, d_2, \ldots, d_n\})^* \longrightarrow V^*,$$

defined by

$$\begin{aligned} h(a) &= h(\bar{a}) = a, \ a \in V, \\ h(c_i) &= h(d_i) = x_i, \ 1 \leq i \leq n. \end{aligned}$$

and

$$g : (V \cup \bar{V} \cup \{c_1, c_2, \ldots, c_n, d_1, d_2, \ldots, d_n\})^* \longrightarrow V^*,$$

defined by

$$\begin{aligned} g(a) &= a, \ g(\bar{a}) = l, \ a \in V, \\ g(c_i) &= x_i, \ g(d_i) = l, \ 1 \leq i \leq n. \end{aligned}$$

Claim 2. *For any language L the following relation holds:*

$$ld_\sigma M(L) = g_M(L) \cup g(h^{-1}(L) \cap L').$$

Proof of the claim. By Lemma 4.3.1, the first term of the above union is the set of all ld_σ-macronuclear strings existing in L and, by the definition of L' and a similar reasoning as in the proof of Proposition 4.3.1, it is easy to infer that the second term is the set of all ld_σ-macronuclear strings obtained as a consequence of the application of ld_σ as many times as possible.

Since the classes of regular and context-free languages are semi AFLs (they are closed under homomorphisms, inverse homomorphisms, intersection with regular sets and union, which implies also the closure under finite transductions), it follows that for any regular/context-free language L, $ld_\sigma M(L)$ is still regular/context-free. These observations complete the proof. □

4.3.2 The *hi* Operation

The *hi* operation works as shown in Figure 4.2. Now, an inverted repeat pair of pointers has been put in evidence. The string is folded into a hairpin aligned by the inverted pair of pointers yielding a new linear string from which a pointer has been dropped.

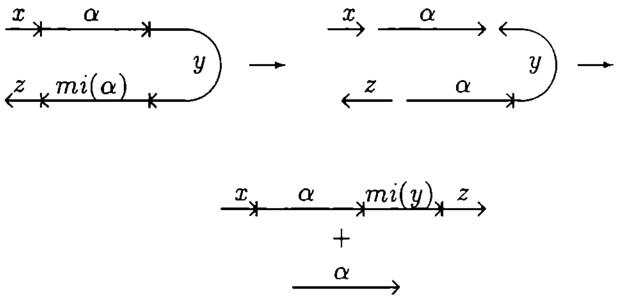

Figure 4.2.

Formally, given a *gac*-scheme $\sigma = (V, P)$, we define

$$hi_\sigma(w) = \{x\alpha mi(y)z \mid w = x\alpha y mi(\alpha)z,\ x, z \in V^*,\ y \in V^+,\ Sub(y) \cap P = \emptyset, \alpha \in P\}.$$

The above operation can be extended to languages in a natural way:

$$hi_\sigma(L) \cup_{w\in L} hi_\sigma(w).$$

A family of languages $\mathcal{F}$ is closed under the operation *hi* if for any language $L \in \mathcal{F}$ and any *gac*-scheme σ, $hi_\sigma(L) \in \mathcal{F}$ holds.

Proposition 4.3.3. *1. The family of regular languages is closed under hi.*

2. The family of context-free languages fails to be closed under hi.

Proof. The first part of the construction is quite similar with the construction in the proof of Proposition 4.3.1. Let $\sigma = (V, P)$ be a *gac*-scheme with $P = \{x_1, x_2, \ldots, x_n,\ mi(x_1),\ mi(x_2),\ \ldots,\ mi(x_n)\}$ for some n. We define the homomorphism

$$h : (V \cup \bar{V} \cup \{c_1, c_2, \ldots, c_n, d_1, d_2, \ldots, d_n\})^* \longrightarrow V^*,$$

defined by

$$\begin{aligned} h(a) &= h(\bar{a}) = a,\ a \in V, \\ h(c_i) &= x_i,\ h(d_i) = mi(x_i),\ 1 \le i \le n. \end{aligned}$$

For a regular language L, now consider the regular languages

$$\begin{aligned} L_1 &= h^{-1}(L) \cap \cup_{i=1}^{n} V^*\{c_i\}(\bar{V}^+ \setminus \bar{V}^*\bar{P}\bar{V}^*)\{d_i\}V^*, \\ L_2 &= h^{-1}(L) \cap \cup_{i=1}^{n} V^*\{d_i\}(\bar{V}^+ \setminus \bar{V}^*\bar{P}\bar{V}^*)\{c_i\}V^*. \end{aligned}$$

The language L_1 can be written as a union of n pairwise disjoint languages

$$L_1 = \cup_{i=1}^{n} L_{1,i}\{c_i\}L_{2,i}\{d_i\}L_{3,i},$$

where all the languages $L_{j,i}$, $1 \le j \le 3$, $1 \le i \le n$, are regular. Let $A = (Q,\ V \cup \bar{V} \cup \{c_1, c_2, \ldots, c_n, d_1, d_2, \ldots, d_n\},\ \delta,\ \{q_0\},\ F)$ be a reduced (without useless states) deterministic finite automaton which recognizes L_1. We consider the following finite automata:

- $A_{i,1} = (Q, V, \delta\,|_{Q\times V}, \{q_0\}, F_{i,1})$, where $F_{i_1} = \{q \in Q \mid$ there exists $s \in Q$ such that $\delta(q, c_i) = s\}$.

- $A_{i,2} = (Q, \bar{V}, \delta\mid_{Q\times\bar{V}}, S_{i,2}, F_{i,2})$, where $S_{i,2} = \{\delta(q, c_i) \mid q \in F_{i,1}\}$ and $F_{i_2} = \{q \in Q \mid$ there exists $s \in Q$ such that $\delta(q, d_i) = s\}$.

- $A_{i,3} = (Q, V, \delta\mid_{Q\times V}, S_{i,3}, F)$, where $S_{i,3} = \{\delta(q, d_i) \mid q \in F_{i,2}\}$.

Since A is minimal, it is easy to check that $L_{j,i}$ is accepted by the automaton $A_{j,i}$ for all $1 \le j \le 3$ and $1 \le i \le n$.

Consequently, the language

$$L_1' = \cup_{i=1}^{n} L_{1,i}\{c_i\}mi(L_{2,i})\{d_i\}L_{3,i}$$

is still regular. Obviously, a similar construction leads to the language L_2' starting from the language L_2.

The reasoning from above for the language L_1 can be schematically illustrated as in Figure 4.3. All edges in the subgraphs denoted by G_0 and G_f are labelled with letters from V while all edges in the subgraphs denoted by $G_1, G_2, \ldots, G_n$ are labelled with letters from $\bar{V}$. Now, we inverse all edges in the subgraphs denoted by $G_1, G_2, \ldots, G_n$. Clearly, the new automaton, which is not necessarily deterministic, accepts the language

$$L_1' = \cup_{i=1}^{n} L_{1,i}\{c_i\}mi(L_{2,i})\{d_i\}L_{3,i}.$$

We now define the homomorphisms

$$g_1, g_2 : (V \cup \bar{V} \cup \{c_1, c_2, \ldots, c_n, d_1, d_2, \ldots, d_n\})^* \longrightarrow V^*,$$

by

$$\begin{aligned} g_1(a) &= g_1(\bar{a}) = g_2(a) = g_2(\bar{a}) = a,\ a \in V, \\ g_1(c_i) &= g_2(d_i) = x_i,\ g_1(d_i) = g_2(c_i) = l,\ 1 \le i \le n. \end{aligned}$$

The equality $g_1(L_1') \cup g_2(L_2') = hi_\sigma(L)$ is immediate. Consequently, the first assertion is proved.

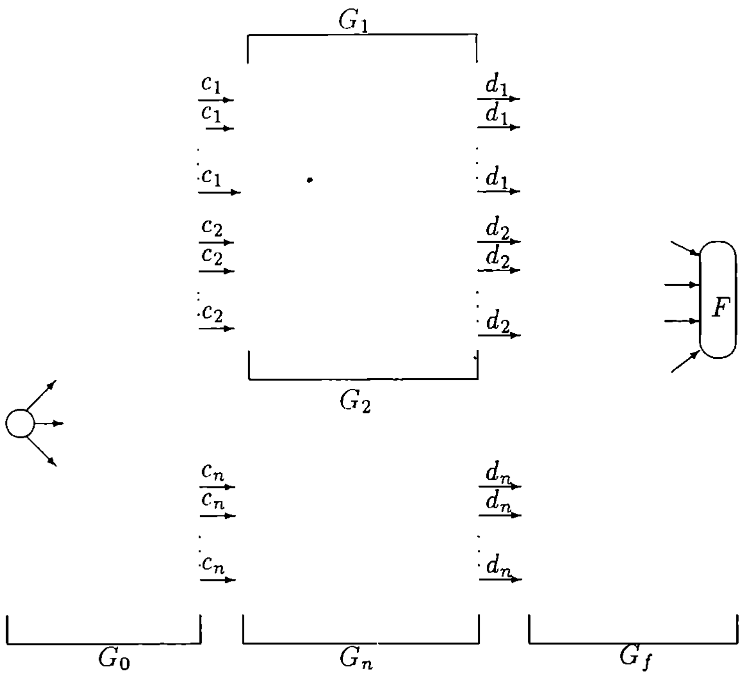

Figure 4.3.

2. It is sufficient to take the linear language

$$L = \{wcmi(w)c \mid w \in \{a,b\}^+\}$$

and the *gac*-scheme $\sigma = (V, \{c\})$. Obviously, $hi_\sigma(L) = \{wcw \mid w \in \{a,b\}^+\}$, which is not context-free. □

It is worth mentioning here that two letter suffice for the previous counterexample. Indeed, we could define the homomorphism h from $\{a,b,c\}$ into the two-letter alphabet $\{a,b\}$ defined by $h(a) = aba, h(b) = abaa$, and $h(c) = bb$. Now, it suffices to take the homomorphical images of all objects defined in the previous counterexample.

4.3.3 The *dlad* Operation

As in the definition of the previous two operations, we explain informally how the *dlad* operation proceeds. This operation is applicable to those strings which have a alternating direct repeat pair of pointers as illustrated in Figure 4.4. This string is folded into two loops each of them aligned by one pair of pointers. The operation removes one occurrence of each pointer and yields a new linear string.

Formally, given a *gac*-scheme $\sigma = (V, P)$ we define

$$dlad_\sigma(w) = \{x\alpha v\beta yuz \mid w = x\alpha u\beta y\alpha v\beta z,\ u,v,y \in V^+,\ x,z \in V^*,\ \alpha,\beta \in P,\ Sub(t) \cap P = \emptyset, t \in \{u,y,v\}\}.$$

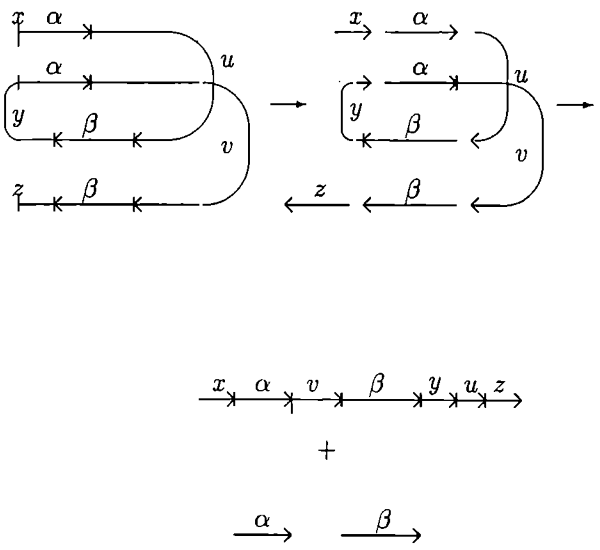

Figure 4.4.

The above operation can be extended to languages in a natural way:

$$dlad_\sigma(L) \cup_{w\in L} dlad_\sigma(w).$$

A family of languages F is closed under the operation *dlad* if for any language $L \in F$ and any *gac*-scheme σ, $dlad_\sigma(L) \in F$ holds.

Proposition 4.3.4. *The class of regular languages is closed under dlad, whereas the class of context-free languages is not closed under this operation.*

Proof. Let $\sigma = (V, P)$ be a *gac*-scheme with $P = \{x_1, x_2, \ldots, x_n\}$ for some n. Consider the new alphabets $V(i,j) = \{a(i,j) \mid a \in V\}$, $1 \leq i,j \leq n$, $i \neq j$. By $P(i,j)$ we denote the set of all strings from P in which each letter a is replaced with $a(i,j)$. We define the homomorphism

$$h : (V \cup (\cup_{i,j=1,i\neq j}^{n} V(i,j)) \cup \{c_1, c_2, \ldots, c_n, d_1, d_2, \ldots, d_n\})^* \longrightarrow V^*,$$

defined by

$$\begin{aligned} h(a) &= h(a(i,j)) = a,\ a \in V, 1 \leq i,j \leq n, i \neq j, \\ h(c_i) &= h(d_i) = x_i,\ 1 \leq i \leq n. \end{aligned}$$

For a regular language L, we now consider the regular language

$$\begin{aligned} L' = {} & h^{-1}(L) \cap \cup_{i,j=1,i\neq j}^{n} V^*\{c_i\}(V^+(i,j) \setminus V^*(i,j)P(i,j)V^* \\ & (i,j))\{c_j\}(V^+(i,j) \setminus V^*(i,j)P(i,j)V^*(i,j))\{d_i\}(V^+(i,j) \setminus V^* \\ & (i,j)P(i,j)V^*(i,j))\{d_j\}V^*, \end{aligned}$$

which can be written (see the proof of Proposition 4.3.3) as a union of $n(n-1)$ pairwise disjoint languages $L_{i,j}$

$$L_{i,j} = L_{i,j,1}\{c_i\}L_{i,j,2}\{c_j\}L_{i,j,3}\{d_i\}L_{i,j,4}\{d_j\}L_{i,j,5},$$

where all the languages $L_{i,j,k}$, $1 \leq i,j \leq n$, $i \neq j$, $1 \leq k \leq 5$, are regular. Clearly, the languages

$$L'_{i,j} = L_{i,j,1}\{c_i\}L_{i,j,4}\{c_j\}L_{i,j,3}\{d_i\}L_{i,j,2}\{d_j\}L_{i,j,5},$$

are regular, too.

We define the homomorphism

$$g : (V \cup (\cup_{i,j=1,i\neq j}^{n} V(i,j)) \cup \{c_1, c_2, \ldots, c_n, d_1, d_2, \ldots, d_n\})^* \longrightarrow V^*,$$

by

$$\begin{aligned} g(a) &= g(a(i,j)) = a,\ a \in V, 1 \leq i,j \leq n, i \neq j, \\ g(c_i) &= x_i,\ g(d_i) = l,\ 1 \leq i \leq n. \end{aligned}$$

The equality

$$g(\cup_{i,j=1,i\neq j}^{n} L'_{i,j}) = dlad_\sigma(L)$$

can be easily checked which concludes the first part of the proposition.

We consider the context-free language

$$L = \{a^n cb^n cbcb^m ca^m \mid n,m \geq 1\}$$

and the *gac*-scheme $\sigma = (\{a,b,c\},\{c\})$. Thus, we get

$$dlad_\sigma(L) = \{a^n cb^m cb^{n+1} a^m \mid n,m \geq 1\}$$

which is not a context-free language. □

Again, the number of letters needed for the last counterexample can be reduced to two by the same homomorphism defined at the end of the previous section.

We now point out two open problems. For a given language L and a *gac*-scheme σ, we can define its hi_σ-macronuclear and $dlad_\sigma$-macronuclear language, respectively, in a similar way to that of defining the ld_σ-macronuclear language. By the results from the previous sections, there exist context-free languages such that the corresponding hi_σ-macronuclear language and the corresponding $dlad_\sigma$-macronuclear language, respectively, is not context-free. We do not know what happens if the given language is regular.

More generally, one may consider the *gac*-macronuclear language of a given language, say L, as the language consisting of all macronuclear strings obtained from the strings of L by applying iteratively the three operations no matter in which order. This problem seems to be fascinating because all three operations are involved in the transformational process of genes in ciliates.

We now continue with a brief discussion about some further directions of research. Let us restrict our discussion to one operation only, say ld, but emphasizing that the problems we discuss here may be addressed to each opeartion as well as to the three operations altogether. For a given string w and a *gac*-scheme σ, we define the ld_σ-macronuclear distance of w as the minimal number of applications of the ld operation to w in order to get a ld_σ-macronuclear string. For a language L, its ld_σ-macronuclear distance is given by the maximal ld_σ-macronuclear distance of its strings, if there is an upper bound or infinite, otherwise. Several problems appear to be of interest with respect to this measure. Is this measure computable for regular languages? What is the complexity of computing this measure for finite languages? Or even for regular languages, if it turns out to be computable?

4.4 Evolutionary Systems

We introduce a language generating device based on string operations suggested by the evolution of cell populations, called evolutionary system. The cells are represented by strings which describe their DNA sequences. The cell community evolves according to gene mutations and divison defined by operations on strings. The paper deals with the generative power of these mechanisms (a characterization of the class of recursively enumerable languages is presented), and the dynamics of the string population. A connection between the growth function of D0L systems and the population growth relation of evolutionary systems is also given.

Description of the dynamics of evolving cell populations is an intriguing question which has been in the focus of interest in current computer science. At the level of individual cells, evolution proceeds by local operations (point mutations) which substitute, insert and delete nucleotides of the DNA sequence. Evolutionary and functional relationships between cells can be captured by taking only local mutations into consideration.

Treating DNA sequences as strings has been often used for investigating the structural information contained in biological sequences. Several approaches have been proposed so far, most of the investigations along these lines deal with grammatical formalisms. The grammatical form is preferable for promoting an abstracted and hierarchical view of the domain. For example, regular grammars have been used for describing very simple genes [11]. Despite one has argued [124] that the genetic language is not more than context-free, these arguments are based on observations restricted just to the amino acid code. Other authors have argued the inadequacy of context-free grammars for modelling the gene regulation [20] or some secondary structures of nucleic acids [121]. In [19] transformational grammars were considered for modelling the gene regulations, while [122] used definite clause grammars, which were constructed on the backbone of context-free grammars with appropriate specifications associated to nonterminals, for investigating gene structure and expression or different forms of mutation and rearrangement. More recently, grammatical formalisms based on string operations suggested by life-like interactions [24] or large scale rearrangements in genomes [33] and [34] have been introduced.

In this paper we present a language generating mechanism, called evolutionary system, inspired by the evolution of cell populations, which might model some properties of evolving cell communities at the syntactical level. We represent cells by strings which describe their DNA sequences. Informally, at any moment of time, the evolutionary system is described by a collection of strings, where each string represents one cell. The cell belongs to species and their community evolves according to mutations and divison which are defined by operations on strings. Only those cells are accepted as survival (correct) ones which are represented by a string in a given set of strings, called the genotype space of the species. This feature parallels with the natural process of evolution.

The first problem discussed in the paper concerns the languages (sets of strings) of the species of the evolutionary systems. We show that any recursively enumerable language is a language of a species of an evolutionary system with point mutations of restricted forms. Then we present results on the dynamics of cell population in the system. We prove that there is no algorithm for deciding whether or not the population of an evolutionary system of regular genotype space is finite, but such an algorithm does exist for systems of finite genotype space. We establish a connection between Lindenmayer systems (language theoretical models of developmental systems) and evolutionary systems by showing that the growth function of any deterministic $0L$ system can be obtained from the population growth relation of some (deterministic) evolutionary system.

Now we proceed to the main definition of the paper, namely that of the *evolutionary system.* In order to illustrate the parallelism between the formal model and the natural process of evolution, we provide each component of the defined system with some informal explanations.

Let n, m be two positive integers. An *evolutionary system* of type (m, n) (an ES, for short) is a construct:

$$\Gamma = (V, S_1, S_2, \ldots, S_m, x_1, x_2, \ldots, x_n),$$

where

- V is the alphabet of the system;
- $S_i = (L_i, M_i, B_i), 1 \leq i \leq m$, are the *species* of the system, with
  - L_i being pairwise disjoint subsets of V^+, called the *genotype space* of the species i,
  - $M_i = (Ins_i, Del_i, Sub_i)$, being the sets of point mutations (*insertions, deletions* and *substitutions*, respectively), where

$$\begin{aligned} Ins_i, Del_i &\subseteq (V \cup \{\varepsilon\}) \times V \times (V \cup \{\varepsilon\}), \\ Sub_i &\subseteq (V \cup \{\varepsilon\}) \times V \times (V \cup \{\varepsilon\}) \times V, \end{aligned}$$

  - B_i being finite subsets of $L_i \times L_i$, called the sets of *division rules.*
- The strings x_j over V, where for each j, $1 \leq j \leq n$, there exists k, $1 \leq k \leq m$, with $x_j \in L_k$, describe the *cells* of the system.

If $Del_i, Ins_i \subseteq \{\varepsilon\} \times V \times \{\varepsilon\}$ and $Sub_i \subseteq \{\varepsilon\} \times V \times \{\varepsilon\} \times V$ for all $1 \leq i \leq m$, then the evolutionary system is called *context-free.*

Evolutionary systems with $L_i \in F$ for all $1 \leq i \leq m$, where F is a class of languages, are called evolutionary systems of F-*genotype* *space.* In the present paper we shall focus our attention on evolutionary systems of finite or regular genotype spaces.

A *configuration* (or a *state*) of an evolutionary system Γ is an element in $(V^+)^r$, for some $r \geq 1$, representing the cells which are present in the system at some moment.
The configuration $(x_1, x_2, \ldots, x_n)$ is said to be the *initial configuration.*

An evolutionary system is functioning by changing its configurations, defined by the direct derivation relation $\Longrightarrow_\Gamma$.

By a direct derivation step a configuration is transformed into another one such that each cell either divides into two offsprings (two new cells) according to a division rule or it evolves into a new cell by some point mutation provided that this new cell belongs to some genotype space. If a cell cannot divide or evolve, it dies disappearing from the new configuration.

Formally, for two configurations $y = (y_1, y_2, \ldots, y_k)$ and $z = (z_1, z_2, \ldots, z_t)$ we define the relation $y \Longrightarrow_\Gamma z$ by the procedure *derivation.*

The reflexive and transitive closure of the relation $\Longrightarrow_\Gamma$ is denoted by $\Longrightarrow_\Gamma^*$. The *language* of the species k, $1 \leq k \leq m$, in an evolutionary system Γ is

$$\begin{aligned} L_k(\Gamma) \quad = \quad & \{y \in L_k | (x_1, x_2, \ldots, x_n) \Longrightarrow_\Gamma^* (y_1, y_2, \ldots, y, \ldots, y_t) \\ & \text{for some } t\}. \end{aligned}$$

That is, the language of species k in the evolutionary system is the set of cells which arise from the initial configuration and belong to the genotype space of species k.

Algorithm 4.4.1 Procedure *derivation(y,z);*

```
begin
t := 0;
for $i := 1$ to $k$ do
   if $y_i \in L_s$, for some $s$ then choose nondeterministically
   a rule $p$ from $B_s \cup Del_s \cup Ins_s \cup Sub_s$;
      if $p = (u, v) \in B_s$ and $y_i = uv$ then
         $t := t + 1$;
```

```
        $z_t := u$;
        $t := t + 1$;
        $z_t := v$;
      endif

  {The cell $y_i$ divides into two offsprings according to a division rule from $B_s$.}

      if ($p = (a, c, b) \in Del_s$) and ($y_i = uacbv$) and ($uabv \in \bigcup_{i=1}^{m} L_i$) then
        $t := t + 1$;
        $z_t := uabv$;
      endif
      if ($p = (a, c, b) \in Ins_s$) and ($y_i = uabv$) and ($uacbv \in \bigcup_{i=1}^{m} L_i$) then
        $t := t + 1$;
        $z_t := uacbv$;
      endif
      if ($p = (a, c, b, d) \in Sub_s$) and ($y_i = uacbv$) and ($uadbv \in \bigcup_{i=1}^{m} L_i$) then
        $t := t + 1$;
        $z_t := uadbv$;
      endif

  { The cell $y_i$ evolves by mutations. Informally, $y_i$ is rewritten
  nondeterministically in one of the three if statements.}

    endif
  endfor
  end.
```

The population of a configuration $(y_1, y_2, \ldots, y_t)$ of an evolutionary system $\Gamma = (V, S_1, S_2, \ldots, S_m, x_1, x_2, \ldots, x_n)$ is defined by

$$\varphi(y_1, y_2, \ldots, y_t) = (q_1, q_2, \ldots, q_m),$$

where $q_i = card\{y_r | y_r \in L_i, 1 \leq r \leq t\}, 1 \leq i \leq m$.

The *population* of Γ is defined by

$$\varphi(\Gamma) = \{\varphi(y_1, y_2, \ldots, y_t) | (x_1, x_2, \ldots, x_n) \Longrightarrow_{\Gamma}^{*} (y_1, y_2, \ldots, y_t)\}.$$

4.4.1 Language of Species

We first show that context-freeness does not diminish the generative power of the evolutionary systems regarding the languages of species. Then we prove that each recursively enumerable language is a language of a species of an evolutionary system of regular genotype space.

Theorem 4.4.1 *For each evolutionary system $\Gamma = (V, S_1, S_2, \ldots, S_m, x_1, x_2, \ldots, x_n)$ there exists a context-free evolutionary system $\Gamma' = (U, S_1', S_2', \ldots, S_m', S_{m+1}, x_1, x_2, \ldots, x_n)$ of the same genotype space as Γ such that $L_k(\Gamma) = L_k(\Gamma')$ for all $1 \leq k \leq m$.*

Proof. Take

$$\begin{aligned} U &= V \cup \{[a,b,c,I] | (a,b,c) \in \bigcup_{k=1}^{m} Ins_k\} \cup \\ & \{[a,b,c,D] | (a,b,c) \in \bigcup_{k=1}^{m} Del_k\} \cup \\ & \cup \{[a,b,c,d] | (a,b,c,d) \in \bigcup_{k=1}^{m} Sub_k\} \end{aligned}$$

and $E = \bigcup_{k=1}^{m} L_k$.

Then the context-free evolutionary system Γ' is defined by

$$\begin{aligned} S_i' &= (L_i, M_i', B_i), \\ M_i' &= (Ins_i', \emptyset, Sub_i'), \\ Ins_i' &= \{(\varepsilon, [a,b,c,I], \varepsilon) | (a,b,c) \in Ins_i\}, \\ Sub_i' &= \{(\varepsilon, b, \varepsilon, [a,b,c,D]) | (a,b,c) \in Del_i\} \cup \{(\varepsilon, b, \varepsilon, [a,b,c,d]) | \\ & (a,b,c,d) \in Sub_i\} \end{aligned}$$

for all $1 \leq i \leq m$ and

$$\begin{aligned} S_{m+1} &= (L_{m+1}, M_{m+1}, \emptyset), \\ L_{m+1} &= Pref(E)\{a[a,b,c,I]c | (a,b,c) \in \bigcup_{k=1}^{m} Ins_k\} Suf(E) \cup \end{aligned}$$

$$
\begin{aligned}
& \cup Pref(E)\{a[a,b,c,D]c|(a,b,c) \in \bigcup_{k=1}^{m} Del_k\}Suf(E) \cup \\
& \cup Pref(E)\{a[a,b,c,d]c|(a,b,c,d) \in \bigcup_{k=1}^{m} Sub_k\}Suf(E), \\
M_{m+1} &= (\emptyset, Del_{m+1}, Sub_{m+1}), \\
Del_{m+1} &= \{(\varepsilon,[a,b,c,D],\varepsilon)|[a,b,c,D] \in U\}, \\
Sub_{m+1} &= \{(\varepsilon,[a,b,c,I],\varepsilon,b)|[a,b,c,I] \in U\} \cup \{(\varepsilon,[a,b,c,d],\varepsilon,d)| \\
& [a,b,c,d] \in U\}.
\end{aligned}
$$

It is easy to notice the double role of the last component of Γ'. It filters the strings obtained by applying the point mutations at wrong sites and restores the other ones. Note that $Pref(E)$, $Suf(E)$ are finite or regular sets provided that E is finite or regular, respectively. □

Obviously, if every species has a finite genotype space, then the language of each species is finite as well. However, regular genotype spaces lead to very complex languages as the next theorem states.

Theorem 4.4.2 *For each recursively enumerable language L there exists an evolutionary system Γ of type (2,1) and of regular genotype space such that $L = L_1(\Gamma)$.*

Proof. Let $L \subseteq V^*$ be a recursively enumerable language generated by a grammar $G = (N, V, S, P)$ in the Geffert normal form [48], namely $N = \{S, A, B, C\}$ and P contains only rules of the form $S \longrightarrow \alpha$, where $\alpha \in (N \cup V)^+$ and $ABC \longrightarrow \varepsilon$. ($N$ denotes the set of nonterminals of G, V is its terminal alphabet, S is the startsymbol and P denotes its production set.)

Put $k = max\{lg(\alpha)|S \longrightarrow \alpha \in P\}$ and let us define an alphabet U by

$$
\begin{aligned}
U &= V \cup \{\$, \#, \#'\} \cup \{[\alpha]|\alpha \in V^+, 1 \leq lg(\alpha) \leq k\} \cup \\
& \{[\alpha]'|\alpha \in V^*, 0 \leq lg(\alpha) < k\},
\end{aligned}
$$

where $\$, \#, \#'$ are new letters not in $N \cup V$.

Now, we consider the evolutionary system

$$\Gamma = (U, S_1, S_2, S),$$

where

$$\begin{aligned} S_1 &= (V^*, M_1, \emptyset),\ M_1 = (\emptyset, \emptyset, \{(\varepsilon, a, \varepsilon, a) | a \in V\}), \\ S_2 &= (U^* \setminus V^*; M_2, \emptyset),\ M_2 = (Ins_2, Del_2, Sub_2) \end{aligned}$$

with

$$\begin{aligned} Ins_2 &= \{(\varepsilon, \$, S), (\varepsilon, \#, A)\} \cup \\ &\quad \{(\$, a, [a\beta]) | a \in V\} \cup \\ &\quad \{(\varepsilon, \$, [\beta]') | 1 \leq lg(\beta) < k\}, \beta \in V^* \end{aligned}$$

$$\begin{aligned} Del_2 &= \{(\varepsilon, \$, \varepsilon), (\varepsilon, \#', \varepsilon), (\#, A, B), (\#, B, C), (\#', C', \varepsilon), \\ &\quad (\varepsilon, [\varepsilon]', \varepsilon)\}, \end{aligned}$$

$$\begin{aligned} Sub_2 &= \{(\#, C, \varepsilon, C'), (\varepsilon, \#, C', \#')\} \cup \\ &\quad \{(\$, S, \varepsilon, [\alpha]) | S \longrightarrow \alpha \in P\} \cup \\ &\quad \{(a, [a\beta], \varepsilon, [\beta]') | 0 \leq lg(\beta) < k, a \in V\} \cup \\ &\quad \{(\$, [\beta]', \varepsilon, [\beta]) | 1 \leq lg(\beta) < k\}, \beta \in V^*. \end{aligned}$$

We state that for each sentential form γ in G, containing at least one nonterminal, there is a derivation $(S) \Longrightarrow^*_{M_2} (\delta)$, with $\delta \in (N \cup V \cup \{\$, \#, \#'\})^*$ such that $\gamma = h(\delta)$, where h is a morphism which erases the symbols $\$, \#, \#'$ and leaves all the other symbols unchanged. We have preferred to use the notation $\Longrightarrow_{M_2}$ instead of $\Longrightarrow_\Gamma$ in order to indicate that all rules used are from M_2.

We prove this assertion by induction on the number of derivation steps made in the generation of γ. The induction basis is obvious. Therefore, it is enough to show how a rule in P, used in the last step of a derivation in G, can be simulated with point mutations in M_2. To begin with, we explain the simulation of the derivation step $\gamma_1 S \gamma_2 \Longrightarrow_G \gamma_1 X_1 X_2 \ldots X_p \gamma_2$.

From the hypothesis of induction, there exists a derivation in Γ of the form $(S) \Longrightarrow_{M_2}^{+} (\delta_1 S \delta_2)$ such that $\gamma_i = h(\delta_i), i = 1,2$. This derivation should be continued as follows:

$$(\delta_1 S \delta_2) \Longrightarrow_{M_2} (\delta_1 \$ S \delta_2) \Longrightarrow_{M_2} (\delta_1 \$ [X_1 X_2 \ldots X_p] \delta_2) \Longrightarrow_{M_2}$$

$$(\delta_1 \$ X_1 [X_1 X_2 \ldots X_p] \delta_2) \Longrightarrow_{M_2} (\delta_1 \$ X_1 [X_2 \ldots X_p]' \delta_2) \Longrightarrow_{M_2}$$

$$(\delta_1 \$ X_1 \$ [X_2 \ldots X_p]' \delta_2) \Longrightarrow_{M_2} (\delta_1 \$ X_1 \$ [X_2 \ldots X_p] \delta_2) \Longrightarrow_{M_2}$$

$$(\delta_1 \$ X_1 \$ X_2 [X_2 \ldots X_p] \delta_2) \Longrightarrow_{M_2} \ldots (\delta_1 \$ X_1 \$ X_2 \$ \ldots$$

$$\$ X_p [\varepsilon]' \delta_2) \Longrightarrow_{M_2} (\delta_1 \$ X_1 \$ X_2 \$ \ldots \$ X_p \delta_2).$$

The dollar signs will be removed by using a rule $(\varepsilon, \$, \varepsilon)$ in Del_2.

Now we consider the simulation of the rule $ABC \longrightarrow \varepsilon$ used in the derivation step $\gamma_1 ABC \gamma_2 \Longrightarrow_G \gamma_1 \gamma_2$. The respective derivation in Γ runs in the following way:

$$(\delta_1 ABC \delta_2) \Longrightarrow_{M_2} (\delta_1 \# ABC \delta_2) \Longrightarrow_{M_2} (\delta_1 \# BC \delta_2) \Longrightarrow_{M_2} (\delta_1 \# C \delta_2)$$

$$\Longrightarrow_{M_2} (\delta_1 \# C' \delta_2) \Longrightarrow_{M_2} (\delta_1 \#' C' \delta_2) \Longrightarrow_{M_2} (\delta_1 \#' \delta_2) \Longrightarrow_{M_2} (\delta_1 \delta_2).$$

Finally, each string in $L_1(\Gamma)$ is obtained from a string in $L_2(\Gamma)$, in which no nonterminal occurs, by removing all occurrences of the symbols $\$, \#, \#'$. When the simulation of the derivation in G is complete, the obtained word, which is a terminal word, is in $L_1(\Gamma)$. Thus, $L(G) \subseteq L_1(\Gamma)$ holds.

Conversely, if $w \in L_1(\Gamma)$, then it is generated by means of rules from M_2. The reverse inclusion, $L_1(\Gamma) \subseteq L(G)$, follows immediately from the fact that for any derivation $(S) \Longrightarrow_{M_2}^{*} (\delta)$ there exists a sentential form γ in G containing at least one nonterminal occurrence such that $\gamma = h(\delta)$, where h is defined above. The proof of this fact can be obtained by a similar reasoning to that previously used. Therefore, $L = L_1(\Gamma)$ holds which concludes the proof. □

As a consequence, by Theorem 4.4.1 we obtain

Theorem 4.4.3 *For each recursively enumerable language L there exists a context-free evolutionary system Γ of type (3,1) and of regular genotype space such that $L = L_1(\Gamma)$.*

It is worth mentioning here that characterizations of recursively enumerable languages based on context-sensitive insertion and deletion operations have been reported in several papers, see, e.g., the chapter devoted to this topic in [104] and the references thereof. Unlike the aforementioned characterizations, Theorem 4.4.3 provides a characterization based on context-free operations but with a control language.

4.4.2 Population of Evolutionary Systems

Theorem 4.4.2 has a series of important consequences regarding the population of the evolutionary systems.

Theorem 4.4.4 *Let Γ be an evolutionary system of regular genotype space. The following problems are undecidable:*

1. *The finiteness of $\varphi(\Gamma)$ i.e. is $\varphi(\Gamma)$ finite?*
2. *Is $\varphi(\Gamma)$ a semilinear set?*
3. *The membership problem for $\varphi(\Gamma)$.*

Proof. We first prove the undecidability of the finiteness of $\varphi(\Gamma)$. To this end, let us consider an arbitrary grammar $G = (N, V, S, P)$ in the Geffert normal form and the following evolutionary system

$$\Gamma = (U \cup \{\&\}, S_1, S_2, S),$$

where U and S_2 are defined in the same way as in the proof of Theorem 4.4.2, & is a new symbol not in U and S_1 consists of

$$\begin{aligned} L_1 &= \{\&, \&\&, ax, \&x, x\}, \text{ for an arbitrary string } ax \\ &\quad \text{with } a \in V, x \in V^+, \\ M_1 &= (\{(\varepsilon, \&, \&)\}, \emptyset, \{(\varepsilon, a, \varepsilon, \&) \cup \{(\varepsilon, b, \varepsilon, b) \mid b \in V\}), \\ B_1 &= \{(\&, x), (\&, \&)\}. \end{aligned}$$

¿From the above construction one can immediately infer that $ax \in L(G)$ if and only if $\varphi(\Gamma)$ is infinite, hence the undecidability status of the finiteness of $\varphi(\Gamma)$ follows.

Moreover, note that $\varphi(\Gamma)$ is a semilinear set if and only if it is finite. Indeed, if $ax \in L(G)$, then we have the following derivation in Γ:

$$(S) \Longrightarrow^+ (ax) \Longrightarrow (\&x) \Longrightarrow (\&, x) \Longrightarrow (\&\&, x) \Longrightarrow (\&, \&, x) \Longrightarrow$$
$$(\&\&, \&\&, x) \Longrightarrow (\&, \&, \&, \&, x) \Longrightarrow^+ (\underbrace{\&, \&, \ldots, \&}_{2^n \text{ times}}, x)$$

for some $n \geq 4$. Consequently, $\varphi(\Gamma) = \{1\} \cup \{2^n + 1 \mid n \geq 0\}$ which is not a semilinear set. As $\varphi(\Gamma)$ is finite provided $ax \notin L(G)$, the second point of the theorem is proved.

A similar construction, left to the reader, can be used in proving the last assertion of the theorem. □

A question that immediately arises is, what one can say about the same problems for evolutionary systems of finite genotype space. We first present a result which will turn out to be a useful tool in giving an answer to this question. The result establishes a connection between evolutionary systems and $0L$ systems (Lindenmayer systems without interactions), which are language theoretic models of developmental systems.

An $0L$ system is a triple $G = (V, s, w)$, where V is an alphabet, s is a finite substitution on V into the set of subsets of V^*, and w is an element of V^*. The language of G is defined by $L(G) = \bigcup_{i \geq 0} s^i(w)$.

Theorem 4.4.5 *For each evolutionary system $\Gamma = (V, S_1, S_2, \ldots, S_m, x_1, x_2, \ldots, x_n)$ of finite genotype space there exists an 0L system G and a morphism h such that $a_1 a_2 \ldots a_p \in L(G)$ if and only if $(x_1, x_2, \ldots, x_n) \Longrightarrow_\Gamma (h(a_1), h(a_2), \ldots, h(a_p))$ holds.*

Proof. Let us consider the alphabet

$$U = \{[x] | x \in \bigcup_{i=1}^{m} L_i\}$$

and the sets

$$A_{[x]} \quad = \quad \{[y] | (x) \Longrightarrow_\Gamma (y) \text{ and } y \in \bigcup_{i=1}^{m} L_i\},$$

$$C_{[x]} = \{[x_1][x_2] | x = x_1x_2, x \in L_j \text{ and } (x_1, x_2) \in B_j, \text{ for some } 1 \leq j \leq m\}$$

for all $[x] \in U$. Then the $0L$ system satisfying the requirements of our theorem is defined by $G = (U, s, [x_1][x_2] \ldots [x_n])$, where the finite substitution s is determined as follows:

$$s([x]) = \begin{cases} A_{[x]} \cup C_{[x]}, \text{ iff } A_{[x]} \cup C_{[x]} \neq \emptyset \\ \{\varepsilon\}, \text{ otherwise.} \end{cases}$$

It is easy to notice that all requirements of our assertion are fulfilled if we choose the morphism h in the following way:

$$h : U^* \longrightarrow V^*, \quad h([x]) = x, \text{ for all } [x] \in U.$$

□

Returning to the decision problems of evolutionary systems of finite genotype space we can state:

Theorem 4.4.6 *Let* Γ *be an evolutionary system of finite genotype space. Then the following problems are decidable:*

1. *The finiteness of* $\varphi(\Gamma)$.

2. *The membership for* $\varphi(\Gamma)$.

Proof. 1. In the previous proof $\varphi(\Gamma)$ is finite if and only if $L(G)$ is finite which is decidable for $0L$ systems.

2. Let $(k_1, k_2, \ldots, k_m)$ be an arbitrarily given m-tuple consisting of nonnegative integers. It belongs to $\varphi(\Gamma)$ if and only if at least one t-tuple $(y_1, y_2, \ldots, y_t)$ can be generated from $(x_1, x_2, \ldots, x_n)$, where

(i) $\quad t = \sum_{j=1}^{m} k_j$

(ii) $\quad k_i$ components in $(y_1, y_2, \ldots, y_t)$ are in L_i for all $1 \leq i \leq m$.

The number of all t-tuples satisfying the aforementioned two conditions is finite which, together with the decidability of the membership problem for $0L$ systems, implies the decidability of the membership problem for $\varphi(\Gamma)$. □

4.4.3 Some Growth Relation Considerations

In this section we discuss evolutionary systems with respect to the growth of their cell populations.

Let $\Gamma = (V, S_1, S_2, \ldots, S_m, x_1, x_2, \ldots, x_n)$ be an evolutionary system. A derivation in Γ consisting of k steps is denoted by $\Longrightarrow_{\Gamma}^{k}$.

The *growth relation* associated to Γ is a function from positive integers into finite subsets of positive integers defined by

$$f_\Gamma(k) = \{p | (x_1, x_2, \ldots, x_n) \Longrightarrow_{\Gamma}^{k} (y_1, y_2, \ldots, y_p)\}.$$

If $card(f_\Gamma(k)) = 1$ for all $k \geq 1$, then f_Γ is called *deterministic* growth relation or a *growth function.*

The growth functions have been very profoundly investigated for $D0L$ systems. By restricting the substitution of a $0L$ system, we obtain a $D0L$ (deterministic $0L$) system. As in the $D0L$ system $G = (V, h, w)$ h is an endomorphism, the derivation process results in a sequence of strings $w_0 = w, w_1 = h(w_0), \ldots, w_n = h(w_{n-1}), \ldots$. The growth function associated to G is defined by $f_G(n) = lg(w_n)$.

The next result establishes a connection between the growth functions of $D0L$ systems and the growth relations of evolutionary systems.

Theorem 4.4.7 *For each $D0L$ system G there exists an evolutionary system Γ with deterministic growth relation and a constant k such that*

$$f_G(p) = f_\Gamma(kp)$$

holds for all $p \geq 0$.

Proof. Let $G = (V, h, w)$ be a $D0L$ system with $V = \{b_1, b_2, \ldots, b_m\}$ and $w = a_1 a_2 \ldots a_n$. Denote by $c(G) = max\{lg(h(a)) | a \in V\}$ and take $k = 3(c(G) + 1)$. Consider

$$U = V \cup \{< b, s > | b \in V, 1 \leq s \leq k - 5\} \cup \{\ll b, s \gg | b \in V,$$

$$0 \leq s \leq k - 6\} \cup \cup\{[x, s] | x \in Suf(h(b)), b \in V, 0 \leq s \leq c(G)\}$$

and construct

$$\Gamma = (U, S_1, S_2, \ldots, S_m, S_{m+1}, a_1, a_2, \ldots, a_n),$$

where the components of each $S_i = (L_i, (Ins_i, Del_i, Sub_i), B_i)$ are defined in the following way:

$$\begin{aligned}
L_i &= \{b_i\} \cup \{< b_i, s > | 1 \leq s \leq k-5\} \cup \{\ll b_i, s \gg | \\
&\quad 0 \leq s \leq k-6\},\\
Ins_i &= Del_i = \emptyset,\\
Sub_i &= \{(\varepsilon, < b_i, s >, \varepsilon, \ll b_i, s-1 \gg) | 1 \leq s \leq k-5\} \cup \\
&\quad \cup\{(\varepsilon, \ll b_i, s \gg, \varepsilon, \ll b_i, s-1 \gg) | 1 \leq s \leq k-6\} \cup \\
&\quad \cup\{(\varepsilon, \ll b_i, 0 \gg, \varepsilon, b_i), (\varepsilon, b_i, \varepsilon, [h(b_i), c(G)])\},\\
B_i &= \emptyset
\end{aligned}$$

for all $1 \leq i \leq m$,

$$\begin{aligned}
L_{m+1} &= \{[x,s] | x \in \bigcup_{b \in V} Suf(h(b)), 0 \leq s \leq c(G)\} \cup \\
&\quad \{< a, 3s-2 > [x, s-1] | a \in V, x \in \bigcup_{b \in V} Suf(h(b)), \\
&\quad 1 \leq s \leq c(G)-1\} \cup \{< a, 3s-2 > [x,s] | a = Pref_1(x), \\
&\quad x \in \bigcup_{b \in V} Suf(h(b)),\ 1 \leq s \leq c(G)\},\\
Ins_{m+1} &= \{(\varepsilon, < a, 3s-2 >, [x,s]) | a = Pref_1(x), \\
&\quad x \in \bigcup_{b \in V} Suf(h(b)), 1 \leq s \leq c(G)\},\\
Del_{m+1} &= \{(\varepsilon, [\varepsilon, s], \varepsilon) | 0 \leq s \leq c(G)\},\\
Sub_{m+1} &= \{(< a, 3s-2 >, [ay, s], \varepsilon, [y, s-1]) | a \in V, \\
&\quad y \in \bigcup_{b \in V} Suf(h(b)), 1 \leq s \leq c(G)\},\\
B_{m=1} &= \{(< a, 3s-2 >, [y, s-1]) | a \in V, y \in \bigcup_{b \in V} Suf(h(b)), \\
&\quad 1 \leq s \leq c(G)\}.
\end{aligned}$$

By induction, one can easily prove that

if $w_p = c_1 c_2 \ldots c_t$, then $(a_1, a_2, \ldots, a_n) \Longrightarrow_{\Gamma}^{kp} (c_1, c_2, \ldots, c_t)$

for all $p \geq 0$. At the same time, for each positive integer p, there is a unique t-tuple $(x_1, x_2, \ldots, x_t)$ such that $(a_1, a_2, \ldots, a_n) \Longrightarrow_{\Gamma}^{p} (x_1, x_2, \ldots, x_t)$. In conclusion, the growth relation associated to Γ is deterministic and $f_G(p) = f_\Gamma(kp)$, for all $p \geq 0$. □

If there exists a polynomial P such that $max(f_\Gamma(k) < P(k))$ for every positive integer k, then f_Γ is *polynomially bounded*; otherwise f_Γ is of *exponential type.*

Corollary 4.4.1 *There are evolutionary systems whose growth relations are polynomially bounded.*

There are evolutionary systems whose growth relations are of exponential type.

Proof. Remember that every $D0L$ growth function is either exponential or polynomially bounded [113]. □

As one might anticipate, it is undecidable whether or not the growth relation of a given evolutionary system of regular genotype space is deterministic. The reader interested in a proof may consult Theorem 4.4.4. As far as the same problem for evolutionary systems of finite genotype space is concerned, we have no complete answer. However, some simple observations can be stated. Given an evolutionary system of finite genotype space Γ and the $0L$ system G constructed in the same way as in the proof of Theorem 4.4.5, denote by

$$\begin{aligned} \Delta &= \{[x] | B_{[x]} \neq \emptyset\}, \\ \Theta &= \{[x] | A_{[x]} \neq \emptyset\}, \\ \Lambda &= \{[x] | s([x]) = \varepsilon\}. \end{aligned}$$

Clearly, $\Lambda \cap \Theta = \Lambda \cap \Delta = \emptyset$. A necessary but not sufficient condition for f_Γ to be deterministic is $\Delta \cap \Theta = \emptyset$. Supposing that $\Delta \cap \Theta = \emptyset$, we state that f_Γ is deterministic iff for every $k \geq 1$ and every $x, y \in s^k(w)$, $2|x|_\Lambda + |x|_\Theta = 2|y|_\Lambda + |y|_\Theta$. The notations used above are w for the axiom of G and $|x|_U$ for the number of all occurrences in x of the symbols in U.

Bibliography

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

[2] Angluin, D: Inductive Inferenceof Formal Languages from Positive Data. *Information and Control* **45** (1980) 117–135

[3] Angluin, D., Smith, C.: Inductive Inference: Theory and Methods. *Computational Surveys* **15** (1983) 237–269

[4] A. Apostolico, F. P. Preparata, Optimal off-line detection of repetitions in a string, *Theoret. Comput. Sci.*, **22** (1983) 297–315.

[5] Arkin, R.C. (Ed.): *Robot Colonies.* Kluwer Academic Publishers, Boston, Mass., 1997

[6] Arkin, R.C.: *Behavior-Based Robotics.* The MIT Press, Cambridge, Mass., 1998

[7] Ashby, W.R.: *An Introduction to Cybernetics.* Chapman & Hall, London, 1961

[8] V. Bafna, P.A. Pevzner, Sorting by transpositions. In *Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms*, 1995.

[9] V. Bafna, P.A. Pevzner, Sorting by reversals: genome rearrangements in plant organelles and evolutionary history of X chromosome. Technical Report CSE-94-032, Department of Computer Science, Pennsylvania State University, 1994.

[10] Boden, M. (Ed.): *The Philosophy of Artificial Life.* Oxford University Press, Oxford, 1996

[11] Brendel, V., Busse, H.G.: Genome structure described by formal languages, *Nucleic Acids Res.* **12** (1984) 2561–2568.

[12] Bro, P.: Chemical reaction automata. *Complexity* **2** (1997) No. 3, 38–44

[13] Brooks, R. A.: *Cambrian Intelligence.* The MIT Press, Cambridge, Mass., 1999

[14] Burks, A. W. (Ed.): *Essays on Cellular Automata.* University of Illinois Press, Urbana, Ill., 1970

[15] Choffrut, C., Karhumäki, J.: Combinatorics on Words. In: [114], Vol. 1, 329–438.

[16] Chomsky, N.: Three models for the description of languages. *IRE Transactions on Information Theory* **IT-2** (1956) 113–124

[17] Chomsky, N.: *Syntactic Structures*, Mouton, The Hague, 1957.

[18] Chomsky, N., Schützenberger, M. P.: The algebraic theory of context-free languages. In: *Computer Programming and Formal Systems* (P. Braffort, D. Hirschberg, eds.). North-Holland, Amsterdam, 1963

[19] Collado-Vides, J.: A transformational-grammar approach to the study of the regulation of gene expression. *J. Theor. Biol.*, 136 (1989), 403–425.

[20] Collado-Vides, J.: The search for grammatical theory of gene regulations is formally justified by showing the inadequacy of context-free grammars, *CABIOS,* 7(1991),321-326.

[21] Copeland, N.G., et al. A genetic linkage map of the mouse: Current applications and future prospects. *Science*, 262(1993), 57–65.

[22] M. Crochemore, An optimal algorithm for computing repetitions in a word, *Inform. Processing Letters*, **12** (1981) 244–250.

[23] Csuhaj-Varjú, E., Dassow, J., Kelemen, J., Paun, Gh.: *Grammar Systems - A Grammatical Approch to Distribution and Cooperation.* Gordon and Breach. Yverdon, 1994

[24] Csuhaj-Varjú, E., Kelemen, J., Kelemenová, A., Paun, Gh.: Eco-grammar systems - a grammatical framework for studying lifelike interactions. *Artificial Life* **3** (1997) 1–28

[25] Csuhaj-Varju, E., Mitrana, V.: Evolutionary systems: A language generating device inspired by evolving communities of cells, *Acta Informatica*, **36** (11), 2000, 913–926.

[26] Culik II, K., Harju, T.: Splicing semigroups of dominoes and DNA, *Discrete Appl. Math.*, **31** (1991), 261 - 277.

[27] Darnell, J., Lodish, H., Baltimore, D.: *Molecular Cell Biology, 2nd Edition.* Scientific American Books, Inc., New York, 1990

[28] Dassow, J.: Grammars with valuations - a discrete model for self-organization of biopolymers. *Discrete Applied Math.* **4** (1982) 161–174.

[29] Dassow, J.: Eine neue Funktion für Lindenmayer-Systeme, *EIK* **12** (1976) 515–521.

[30] Dassow, J., Mitrana, V.: Splicing grammar systems, *Computers and AI* **15** (1996) 2-3, 109–122.

[31] Dassow, J., Paun, Gh., Rozenberg, G.: Grammar systems. In:*Handbook of Formal Languages vol. 2.* (G. Rozenberg, A. Salomaa, eds.). Springer-Verlag, Berlin, 1997, 155–213

[32] Dassow, J., Mitrana, V.: On some operations suggested by genome evolution. In: *Proc. Second Pacific Symposium on Biocomputing* (R. B. Altman, A. K. Dunker, L. Hunter, T. Klein, eds.), World Scientific, 1997, 97–108.

[33] Dassow, J., Mitrana, V.: Evolutionary grammars: a grammatical model for genome evolution. In: *Bioinformatics* (Proc. German Conference on Bioinformatics, R. Hofestädt, Th. Lengauer, M. Löffler, D. Scomburg, eds.), Lectures Notes in Comp. Science 1278, Springer Verlag, 199–209.

[34] Dassow, J., Mitrana, V., Salomaa, A.: Context-free evolutionary grammars and the structural language of nucleic acids, *BioSystems* **43** (1997), 169–177.

[35] Dassow, J., Mitrana, V.: Self cros-over systems. *Computing with Bio-Molecules* (Gh. Păun ed.), Springer-Verlag Singapore, 1998, 283–294.

[36] Dassow, J., Păun, Gh.: Remarks on operations suggested by mutations in genomes, *Fundam. Inform.* **36** (1998) 183–200.

[37] Dassow, J., Păun, Gh.: On the regularity of languages generated by context-free evolutionary grammars. To appear in *Discrete Appl. Math.*

[38] Dassow, J., Mitrana, V., Păun), Gh.: Point mutations in context-free languages. *Proc. of DLT'97* (S. Bozapalidis ed.), 1997, 429–446.

[39] Dassow, J., Mitrana, M., Păun, Gh.: On the regularity of duplication closure, *Bull. EATCS*, **69** October 1999, 133–137.

[40] Dassow, J.: Numerical parameters of evolutionary grammars. In: J. Karhumäki, H. Maurer, Gh. Păun, G. Rozenberg, eds., *Jewels are Forever*, Springer-Verlag, Berlin, 1999, 171–181.

[41] Dennett, D.: Artificial life as philosophy.*Artificial Life* **1** (1994) 291–292

[42] Ehrenfeucht, A., Prescott, D.M., Rozenberg, G.: Computational aspects of gene (u)scrambling in ciliates. In *Evolution as Computation*, (L. Landwerber, E. Winfree, eds.), Springer Verlag, Berlin, 2000, 45–86.

[43] Ehrenfeucht, A., Petre, I., Prescott, D.M., Rozenberg, G.: Universal and simple operations for gene assembly in ciliates. In *Where Mathematics, Computer Science, Lingusitics and Biology Meet*, (C. Martin-Vide, V. Mitrana, eds.), Kluwer Academic, Dordrecht, 2000, 329–343.

[44] Farmer, J. D., d'A Belin, A: Artificial life: the coming development. In: *Artificial Life* **II.** (Ch. G. Langton et al., eds.). Addison-Wesley, Redwood City, Cal., 1991, 815–840

[45] Freund, R., Martin-Vide, C., Mitrana, V.: On some operations suggested by gene assembly in ciliates (submitted).

[46] Fu, S. K.: *Syntactic Methods in Patters Recognition.* Academic Press, New York, 1974

[47] A. Gabrielian, Pure grammars and pure languages, Univ. of Waterloo, Dept. of Computer Research Report CSRR 2027, 1970.

[48] Geffert, V.: Normal forms for phrase-structure grammars. *RAIRO/Theoretical Informatics and Applications*, **25, 5** (1991), 473–496.

[49] McGeoch, D.J.: Molecular evolution of large DNA viruses of eukaryotes. *Seminars in Virology*, **3** (1992), 399–408.

[50] Ginsburg, S., Spanier, E.H.: Bounded regular sets. *Proc. Amer. Math. Soc.* **17** (1966) 1043–1049

[51] Ginsburg, S.: *An Introduction to Mathematical Machine Theory.* Addison-Wesley, Reading, Mass., 1962

[52] Ginsburg, S.: *The Mathematical Theory of Context-Free Languages.* McGraw Hill, New York, 1966

[53] Grate, L. et al.: RNA modelling using Gibbs sampling and stochastic context-free grammars. In *Proc. of Second Int. Conf. on Intelligent Systems for Molecular Biology*, Menlo Park, CA, August 1994, AAAI/MIT Press.

[54] Gruska, J.: Descriptional complexity of context-free languages. In: *Proc. Math. Found. Comp. Science*, 1973, 71–83.

[55] S. Hannenhalli et al. Algorithms for genome rearrangements: herpesvirus evolution as a test case. In *Proc. of the 3rd International Conference on Bioinformatics and Complex Genome Analysis*, 1994.

[56] Harrison, M. A.: *Introduction to the Formal Language Theory.* Addison-Wesley, Reading, Mass., 1978

[57] Hartl, D.L., Freifelder, D., Snyder, L.A.: *Basic Genetics*, Jones and Bartlett Publ., Boston, Portola Valley, 1988.

[58] Hartmanis, J.: On weight of computations. *EATCS Bulletin* **55** (1995) 136–138.

[59] Head, T.: Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviour. *Bull. Math. Biology* **49** (1987) 737–759.

[60] Head, T., Păun, Gh., Pixton, D.: Language theory and molecular genetics. In: [114], Vol. 2, 295–360.

[61] Herman, G., Rozenberg, G.: *Developmental Systems and Languages.* North-Holland, Amsterdam, 1975

[62] Holland, J. H.: *Emergence - From Chaos to Order.* Addison-Wesley, Reading, Mass., 1998

[63] Hopcroft, J., Ullman, J. D.: *Formal Languages and Their Relation to Automata.* Addison-Wesley, Reading, Mass., 1969

[64] Hopcroft, J.E., Ullman, J.D.: *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Publ. Co., Reading, 1979.

[65] Ilie, L., Mitrana, V.: Crossing-over on languages. A formal representation of the recombination of genes in a chromosome, Submitted, 1995.

[66] J. Kececioglu, D. Sankoff, Exact and approximation algorithms for sorting by reversals, with application to genome rearrangements. In *Proc. of the 4th Symposium on Combinatorial Pattern Matching*, Springer-Verlag, LNCS 684, 1993, 87–105.

[67] J. Kececioglu, D. Sankoff, Efficient bounds for oriented chromosome-inversion distance. In *Proc. of the 5th Symposium on Combinatorial Pattern Matching*, Springer-Verlag, LNCS 807, 1994, 307–325.

[68] J. Kececioglu, R. Ravi, Of mice and men: Algorithms for evolutionary distances between genomes with translocation. In *Proceedings of the 6th ACM-SIAM Symposium on Discrete Algorithms*, 1995, 604–613.

[69] Knuth, D.E., Morris, J.H., Pratt, V.R.: Fast pattern matching in strings, *SIAM J. Comput.* **6** (1977), 323–350.

[70] Kari, L.: *On Insertion and Deletion in Formal Languages*, PhD Thesis, University of Turku, 1991.

[71] Karlin, S., Mocarski, E.S., Schachtel, G.A.: Molecular evolution of herpesviruses: genomic and protein comparisons. *J. of Virology*, **68**(1994),1886–1902.

[72] Kelemenová, A., Kelemen, J.: An interview with A. Lindenmayer.*Bulletin of the EATCS* **23** (1984) 185–198

[73] Kelemen, J., Kelemenova, A., Martin-Vide, C., Mitrana, V.: Colonies with limited activation of components, *Theoretical Computer Science*, **244, 1-2** (2000), 289–298.

[74] Kleene, S. C.: Representation of events in nerve nets and finite automata. In: *Automata Studies* (C. E. Shannon, J. McCarthy, eds.). Princeton University Press, Princeton, 1956, 1–41

[75] Kobayashi, S., Mitrana, V., Păun, Gh., Rozenberg, G.: Formal properties of PA-Matching, *Theoretical Computer Science* 2000 (in press).

[76] Landweber, L.F., Kari, L.: The evolution of cellular computing: nature's solution to a computational problem, *Proc. of the*

4th DIMACS Meeting on DNA Based Computers, Philadelphia, 1998, 3–15.

[77] Landweber, L.F., Kari, L.: Universal molecular computation in ciliates, *Evolution as Computation*, (L. Landwerber, E. Winfree, eds.), Springer Verlag, Berlin, to appear.

[78] Langton, Ch. G.: Artificial life. In: *Artificial Life, Proc. Interdisciplinary Workshop on Synthesis and Simulation of Living Systems.* (Ch. G. Langton, Ed.). Addison-Wesley, Redwood City, Cal., 1989, 1–47

[79] Langton, Ch. G. (Ed.): *Artificial Life - An Introduction.* The MIT Press, Cambridge, Mass., 1995

[80] Ledlay, R. S.: High-speed automatic analysis of biomedical pictures. *Science* **146** (1964) 216–223

[81] Ledlay, R. S., Rotolo, L. S., Golab, T. J., Jacobsen, J. D., Ginsburg, M. D., Wilson, J. B.: FIDAC - film input to digital automatic computer and associated syntax-driven pattern recognition programming system. In: *Optical and Electro-Optical Information Processing.* (J. T. Tipplett et al., eds.). The MIT Press, Cambridge, Mass., 1965, 591–614

[82] Lindenmayer, A.: Life cycles as hierarchical relations. In: *Form and Strategy in Science.* (J. R. Gregg and F. T. C Harris, eds.). Reidel, Dordrecht, 1964, 416–470

[83] Lindenmayer, A.: Mathematical models of cellular interactions in development, I. Filaments with one-sided inputs, II. Simple and branching filaments with two-sided inputs. *Journal of Theoretical Biology* **18** (1968) 280–299, 300–315

[84] M. G. Main, R. J. Lorentz, An $O(n \log n)$ algorithm for finding all repetitions in a string, *J. of Algorithms*, 1984.

[85] M. G. Main, R. J. Lorentz, Linear time recognition of squarefree strings, in (A. Apostolico, Z. Galil, eds.) *Combinatorial Algorithms on Words*, Springer, 1985, 271–278.

[86] Manaster Ramer, A.: Uses and misuses of mathematics in linguistics, *Proc. X Congreso de Lenguajes Naturales y Lenguajes Formales*, Sevilla, 1994.

[87] Marcus, S.: Linguistics as a pilot science. In *Current Trends in Linguistics*, Th. Sebeok, vol. 12, Mouton, The Hague, 1974.

[88] Marcus, S.: Linguistic structures and generative devices in molecular genetics, *Cahiers de Linguistique Théorique et Appliquée*, **11, 1** (1974), 77–104.

[89] Martin-Vide, C., Păun, Gh.: Duplication grammars, *Acta Cybernetica* **14** (1999) 151–164.

[90] Martin-Vide, C., Mitrana, V.: On context-free duplications, *Recent topics in mathematical and computational linguistics*, (C. Martin-Vide, Gh. Păun, eds.), The Publishing House of the Romanian Academy, 2000, 196–206.

[91] Maurer, H.A., Salomaa, A., Wood, D.: Pure grammars,' *Information and Control*, **44** (1980) 47–72.

[92] Ming-wei Wang, On the irregularity of the duplication closure, *EATCS Bulletin*, **70** (2000), 162–163.

[93] Mitrana, V.: Crossover systems. A generalization of splicing systems, *J. of Automata, Languages and Combinatorics*, **2**(1997), 151–160.

[94] Mitrana, V., Rozenberg, G.: Some properties of duplication grammars, *Acta Cybernetica* **14** (1999) 165–177.

[95] V. Mitrana, G. Rozenberg, A. Salomaa, On the crossover distance, (submitted).

[96] Maurer, H.A., Salomaa, A., Wood, D.: Pure grammars, *Information and Control*, **44** (1980) 47–72.

[97] von Neumann, J.: *Theory of Self-Reproducing Automata.* University of Illinois Press, Urbana, Ill., 1966

[98] Newell, A.: *Unified Theories of Cognition.* Harvard University Press, Cambridge, Mass., 1990

[99] Newell, A., Simon, H. A.: *Human Problem Solving.* Prentice-Hall, Englewood Cliffs, NJ., 1972

[100] Palmer, J.D., Herbon, L.A.: Plant mitochondrial DNA evolves rapidly in structure, but slowly in sequence. *Journal of Molecular Evolution*, **27** (1988), 87–97.

[101] Paun, Gh. (Ed.): *Artificial Life - Grammatical Models.* The Black See University Press, Bucharest, 1995

[102] Păun, Gh.: Regular extended H systems are computationally universal, *J. Automata, Languages, Combinatorics*, **1**, **1** (1996), 27 - 36.

[103] Paun, Gh. (Ed.): *Computing with Bio-Molecules - Theory and Experiments.* Springer-Verlag, Singapore, 1998

[104] Paun, Gh., Rozenberg, G., Salomaa, A.: *DNA Computing - New Computational Paradigms.* Springer-Verlag, Berlin, 1998

[105] Păun, Gh.: On the splicing operation, *Discrete Applied Mathematics*, to appear.

[106] Prescott, D.M.: Cutting, splicing, reordering, and elimination of DNA sequences in hypotrichous ciliates, *BioEssays*, **14**, **5** (1992), 317–324.

[107] Prusinkiewicz, P., Lindenmayer, A.: *The Algorithmic Beauty of Plants.* Springer-Verlag, New York, 1990

[108] M. O. Rabin, Discovering repetitions in strings, in (A. Apostolico, Z. Galil, eds.) *Combinatorial Algorithms on Words*, Springer, 1985, 279–288.

[109] Reif, J.H.: Parallel molecular computation: Models and simulations, *Proc. of Seventh Annual ACM Symp. on Parallel Algorithms and Architectures*, Santa Barbara, 1995, 213 - 223

[110] Rounds, W.C., Manaster Ramer, A., Friedman, J.: Finding natural languages a home in formal language theory. In *Mathematics of Language* (A. Manaster Ramer ed.), John Benjamins, Amsterdam, 1987, 349–360.

[111] Rozenberg, G., Salomaa, A. (eds.): *L Systems.* Springer-Verlag, Berlin, 1974

[112] Rozenberg, G., Salomaa, A. (eds.): *The Book of L.* Springer-Verlag, Berlin, 1986

[113] Rozenberg, G., Salomaa, A.: *The Mathematical Theory of L Systems.* Academic Press, 1980.

[114] Rozenberg, G., Salomaa, A. (Eds.): *Handbook of Formal Languages*, vol. I-III, Springer, Berlin, 1997.

[115] Sakakibara, Y.: *An Efficient Learning of Context-Free Grammars from Positive Structural Examples.* Intern. Inst. for Ad-

vanced Study of Social Inform. Sci. Research Report 93, Numazu, 1989

[116] Sakakibara, Y. et al., Stochastic context-free grammars for t-RNA modelling, *Nucleic Acids Research*, **25** (1994) 5112–5120.

[117] Salomaa, A.: *Formal Languages.* Academic Press, New York, 1973

[118] Salomaa, A.: *Jewels of Formal Languages.* Computer Science Press, Rockville, 1981.

[119] Sankoff, D.: Edit distance for genome comparison based on non-local operations. In *Proc. of the 3rd Symposium on Combinatorial Pattern Matching*, Springer-Verlag, LNCS 644, 121–135, 1992.

[120] Sankoff, D. et al. Gene order comparisons for phylogenetic inference: Evolution of the mitochondrial genome. *Proc. Natl. Acad. Sci. USA*, **89**(1992),6575–6579.

[121] Searls, D.B.: The computational linguistics of biological sequences. In *Artificial Intelligence and Molecular Biology* (L. Hunter ed.), AAAI Press, The MIT Press, 1993, 47–120.

[122] Searls, D.B.: Formal grammars for intermolecular structure. In *IEEE Symp. on Intelligence in Neural and Biological Systems*, IEEE Computer Society Press, 1995, 30–37.

[123] Searls, D.B.: String variable grammar: A logic grammar formalism for the biological language of DNA, *Journal of Logic Programming*, in press.

[124] Shanon, B.: The genetic code and human language, *Synthese*, **39** (1978), 401–415.

[125] Shyr, H.J.: *Free Monoids and Languages.* Inst. Applied Math., Univ. Taichung and Hon Min Book Co., Taichung, Taiwan.

[126] Simovici, D. A., Tenney, R. L.: *Theory of Formal Languages with Applications.* World Scientific, Singapore, 1999

[127] Therman, E., Susman, M.: *Human Chromosomes, Structure, Behavior, and Effects.* Springer-Verlag, 1993.

[128] Thue, A.: Uber unendliche Zeitchenreihen. *Norske Vid. Selsk. Skr. I. Mat. Nat. Kl. Christiania* 7 (1906) 1–22.

[129] A. Thue, Uber die gegenseitige Lage gleicher Teile gewiisser Zeichenreihen, *Norske Videnskabers Selskabs Skrifter Mat.-Nat. Kl. (Kristiania)*, 1 (1912) 1–67.

[130] Walter, G.: *The Living Brain.* Norton & Co., New York, 1953

[131] Watson, J., D.: *The Double Helix - A Personal Account of the Discovery of the Structure of DNA.* Weidenfeld and Nicolson, London, 1968

[132] Wilson, S. W.: The animate path to AI. In: *From Animals to Animates.* (J.-A. Meyer et al., eds.). Cambridge, Mass.: The MIT Press, 1991, 15–21

[133] Woodger, J. H.: *The Axiomatic Method in Biology.* Cambridge University Press, Cambridge, 1937

[134] Yokomori, T, Kobayashi, S.: DNA evolutionary linguistics and RNA structure modelling: a computational approach. In *IEEE Symp. on Intelligence in Neural and Biological Systems*, IEEE Computer Society Press, 1995, 38–45.

Tiparul s-a executat sub cda 798/2001
la Tipografia Editurii Universității din Bucureşti

65.500 le

ISBN: 973–575–535–1

Lei 65500