

Institut für Informatik
Universität Potsdam

**Contributions to the Syntactical Analysis
Beyond Context-Freeness**

Habilitationsschrift

zur Erlangung des akademischen Grades

“*doctor rerum naturalium habilitatus*”

(Dr. rer. nat. habil.)

in der Wissenschaftsdisziplin “Theoretische Informatik”

eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät
der Universität Potsdam

von
Henning Bordihn

Tag des Kolloquiums: 17. Februar 2012

Published online at the
Institutional Repository of the University of Potsdam:
URL <http://opus.kobv.de/ubp/volltexte/2012/5971/>
URN <urn:nbn:de:kobv:517-opus-59719>
<http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-59719>

Preface

The theory of formal grammars and languages has its origins in the fifties of the 20th century, when Noam Chomsky suggested a classification of formal grammars describing the syntax of languages [27]. Since then, this theory has been developed by many researchers into one of the theoretical fundamentals of computer science. Apart from its linguistic roots, this development has been strongly influenced by the fields of combinatorics and algebra of semigroups and monoids as well as the theory of computing.

Comprehensively developed parts of formal language theory allow applications such as language processing. In particular, the rise of high-level programming languages required a theoretically sound methodology for constructing compilers. Here and in other language processing applications, the *syntactical analysis (parsing)* is one of the main application areas of formal language theory based on formal grammars.

In connection with programming languages, context-free grammars (one of the classes suggested by Chomsky) have been established as a useful formalism for describing the syntax of languages, which led to the remarkable development of a theory of parsing of context-free languages. However, many application areas of formal languages exhibit inherent aspects that cannot be described with the help of context-free grammars. Examples of such areas are, besides linguistics, developmental biology, logic, graph theory and so on; for a detailed discussion see [41]. The grammars of the next level of the classification given by Chomsky, namely the context-sensitive grammars, turned out to be too powerful for efficient applications and are rather lacking a coherent interpretation in the fields of application. Therefore, a large number of grammar formalisms has been introduced whose generative power is in between that of context-free and context-sensitive grammars. It has been the intention to obtain mechanisms which are both powerful enough for the description of all syntactical aspects of the applications and preserve as much as possible of the simplicity of context-free grammars.

Although those “intermediate” mechanisms have been thoroughly investigated, there is hardly any application which is in line with their motivation. The main reason for this might be the fact that an important property of context-free grammars is missing in most cases, namely that of allowing an efficient syntactical analysis. The presented thesis aims at attacking this problem. It systematically surveys results obtained by the author together with his co-authors which can be viewed as contributions to the area of syntactical analysis of non-context-free languages.

In the first chapter the motivation, as sketched above, is elaborated. Moreover, the basic notions and notation used throughout the thesis are presented.

The second chapter provides a survey of non-context-free grammar formalisms treated in this thesis, reviewing their definitions and results with respect to their capacities when describing languages.

In Chapter 3, these grammar formalisms are treated as accepting devices, working as

language recognizers and thus—possibly—as syntactical analyzers. It is investigated in which cases the accepting feature adds to the power of the underlying grammar formalisms and which families of languages can be analyzed. Furthermore, an application of the concept of accepting grammars to the field of computational complexity is presented by giving several grammatical characterizations of an unsolved complexity problem, namely the LBA problem.

Another approach to the syntactical analysis of languages described by non-context-free grammar formalisms is taken in Chapters 4 and 5. Using the example of cooperating distributed grammar systems, it is shown how non-context-free grammar formalisms can be restricted so that efficient parsing becomes possible. The class of grammar systems obtained by those restrictions is investigated with respect to several properties, and an efficient parsing algorithm is presented.

It should be mentioned that this thesis is not intended to be a collection of parsing algorithms for non-context-free grammar formalisms, and it is deliberately not restricted to efficient syntactical analysis. It surveys several different *contributions to the topic*, including theoretical investigations of the families of languages which can be analyzed by the mechanisms treated in the thesis.

Most of the results presented here have been published in scientific journals or conference proceedings, in particular the results

- from Section 3.1.1 in [17],
- from Section 3.1.2 in [15, 16, 17, 52],
- from Section 3.1.3 in [11, 53],
- from Section 3.1.4 in [13, 14],
- from Section 3.2 in [21] after a preliminary version [8],
- from Chapter 4 in [25] after an extended abstract [23],
- from Chapter 5 in [24] after a preliminary but more comprehensive version in [22].

Theorems, lemmata and corollaries of this thesis are given together with references to the articles where they have been published, if applicable.

The author would like to express his gratitude to his co-authors of the joint papers which have become the basis of this thesis, that is, Erzsébet Csuhaj-Varjú, Jürgen Dassow, Henning Fernau, Markus Holzer and György Vaszil. It is a need to mention that Jürgen Dassow suggested the topics of formal languages and syntactical analysis to the author and was frequently available for helpful discussions. The author is also grateful to Helmut Jürgensen for his continual support and patient encouragement to complete this thesis. Last but not least, the author wishes to thank his family, in particular his wife Antje and daughter Inga, who had to endure the plenty of time the author spent working on this thesis.

Contents

1	Introduction	7
1.1	Syntactical Analysis and Language Processing	7
1.2	Basic Definitions and Notation	10
1.2.1	Languages, Language Operations, Families of Languages	10
1.2.2	Automata, Grammars and the Chomsky Hierarchy	12
1.2.3	Basic Elements of the Theory of Parsing	15
1.3	Insufficiency of Context-Freeness	17
2	Non-Context-Free Grammar Formalisms	21
2.1	Definitions	23
2.2	Generative Capacities	30
2.2.1	Single Devices: The Non-Erasing Case	30
2.2.2	Single Devices: The Erasing Case	31
2.2.3	Grammar Systems	33
3	Accepting Grammars and Systems	34
3.1	Accepting versus Generating Mode	35
3.1.1	Context Condition Grammars	35
3.1.2	When cc Grammars Do Not Help	39
3.1.3	Cooperating Distributed Grammar Systems	45
3.1.4	Parallel Communicating Grammar Systems	53
3.2	Accepting Grammars and the LBA Problem	64
3.2.1	Restricting Nondeterminism in Programmed Grammars	65
3.2.2	On Context-Sensitive and Deterministic Context-Sensitive Languages	73
3.2.3	Programmed Grammars with Regulation	78
3.3	Discussion	79
4	Leftmost Derivations	82
4.1	Definitions	83
4.2	The Power of Leftmost Derivations in CD Grammar Systems	85
4.2.1	The So-Called Trivial Derivation Modes	85
4.2.2	The Terminal Mode of Derivation	86
4.2.3	The $=k$ and $\geq k$ Modes of Derivation	87
4.2.4	The Full Competence Mode of Derivation	92
4.3	Conclusion	94

5	Deterministic Parsing for CD Grammar Systems	96
5.1	Definitions	97
5.2	On the Power of $LL(k)$ CD Grammar Systems	99
5.3	Using Lookup Tables	102
5.4	Conclusion	113
	Bibliography	114

Chapter 1

Introduction

1.1 Syntactical Analysis and Language Processing

The core of most language processing devices is the mapping of one representation of certain structures into another. Here, language is regarded as a concept in the broader sense, including natural languages, programming languages, languages of graphical representations, sets of DNA strands as languages encoding genetical information, and so forth. Typical examples of language processing devices are automated language translators for natural languages, compilers, code generation or virtualization tools, parts of DNA analyzing tools etc. Natural languages can be translated into another natural language or into machine language, including translations by speech recognition devices. Compilers are needed in order to translate high-level programming languages into code executable by computer processors or virtual machines. In modern integrated (software) development environments (IDEs), portions of compilers continuously check the correctness of the code under development, indicating errors or violations of certain conventions. In the context of model driven software development (see, for example, [101] or [73] for the approach of the Object Modelling Group), tools for modelling the structure or behaviour of the software to be developed gain more and more interest. In order to circumvent a loss of information when stepping from those models to technical realizations, model transformation of (frequently graphical) models to executable code are of crucial importance. Code virtualization can be viewed as converse translation, which may help to understand and maintain the business logic contained in operating software products, what may come in useful when rigorous documentation is missing and software maintenance or re-engineering is needed.

All these formal language applications have one feature in common: sentences of a language representing some information or structure is transformed into sentences of another language representing the same pieces of information or structure. A compiler, for example, consists of several portions, each realizing such transformation. The translation performed by the compiler as a whole is the composition of the mappings executed by the single portions. The main phases of a compiler and their main tasks are

1. the *lexical analysis* in which the source code of a program as sequence of, say, ASCII or Unicode symbols is translated into a sequence of *tokens* representing reserved key words of the programming language, identifiers, literals and operators, punctuation symbols, and so forth;

2. the *syntactical analysis* in which the sequence of tokens is checked against the rules specifying the syntax of the programming language (that is, the grammar, frequently given in Backus-Naur Form (BNF)¹) and, in the affirmative case, the *syntax tree* is produced reflecting the syntactical rules which are applied in order to obtain the given program;
3. the *weed* in which the syntax tree is transformed into a reduced form, called *abstract syntax tree*, ridded of redundant or unnecessary parts such that it is more suitable for the processing in further phases;
4. the *code generation* yielding machine code or, as a preliminary step, assembler code or code executable by a virtual machine;
5. the *code optimization* aiming to make the code more compact and efficient when executed.

Each phase takes the result of its preceding phase as input, translating it to a new representation as output. Nevertheless, it should be noted that this sequence of phases is a very simplified description of a compiler. At first, some important parts are not listed here, such as the symbol processing (scoping), type checking and further static checks of additional requirements on the syntax of a program, for example, in order to make sure in languages like Java that each local variable has been initialized prior to any reading access to it. At second, one should be aware of the fact that a real compiler does not sequentially run through the phases as listed above. The phases, together with the symbol processing, type checking and further static analysis, are rather nested and interleaved, see [5, 102].

Other language processing devices consist of similar but application specific phases. One principal phase is fundamental for all those processes when sentences of languages (specified by some grammar) are translated into sentences of another language (specified by another grammar), namely the syntactic analysis. A sentence of a given language (a sentence of a natural language, a program of a programming language etc.) as a mere sequence of symbols is insufficient for the purpose of language processing. During the syntactical analysis, the membership of the sentence in the language is checked. The sentence is a member of the language if it can be constructed according to the grammatical rules of the language. If so, then a sequence of rules is derived in which the given sentence can be constructed (depending on the construction scheme). Such sequence is said to be a *parse* of the sentence. The parse can be graphically represented as syntax tree; it reveals the grammatical structure of the sentence which is particularly needed for further language processing steps. Consider, for example, the sentence of the natural language English:

The man writes the letter.

The (a little bit simplified) grammatical rules according to which this sentence is constructed can be written in Backus-Naur Form as follows.

- ```
(r1) <sentence> ::= <noun phrase><verb phrase>
(r2) <noun phrase> ::= <determiner><noun>
```

---

<sup>1</sup>The Backus-Naur Form of grammatical rules is a unified representation of *context-free rules* which are formally introduced in Section 1.2. For examples, see [3, 102].



(r3)  $\langle \text{verb phrase} \rangle ::= \langle \text{verb} \rangle \langle \text{noun phrase} \rangle$

Then, one parse of the sentence is  $r1r2r3r2$ , expressing that the *sentence* consists of a *noun phrase* followed by a *verb phrase* (r1), next the noun phrase is a *determiner* followed by a *noun* (r2), the *verb phrase* is a sequence of a *verb* and a *noun phrase* (r3), the latter of which is developed according to (r2) again. Note that, depending on the order in which grammatical categories are replaced according to the rules, also another parse can be found, namely  $r1r3r2r2$ . In both cases, the syntax tree built on the sentence is given in Figure 1.1. Here,  $\langle s \rangle$ ,  $\langle np \rangle$ ,  $\langle vp \rangle$  and  $\langle det \rangle$  are abbreviatory written for  $\langle \text{sentence} \rangle$ ,  $\langle \text{noun phrase} \rangle$ ,  $\langle \text{verb phrase} \rangle$  and  $\langle \text{determiner} \rangle$ , respectively. In such syntax tree, a “top-down walk” from the root (here  $\langle s \rangle$ ) to the leaves corresponds to the rule applications which yield the sentence.<sup>2</sup>

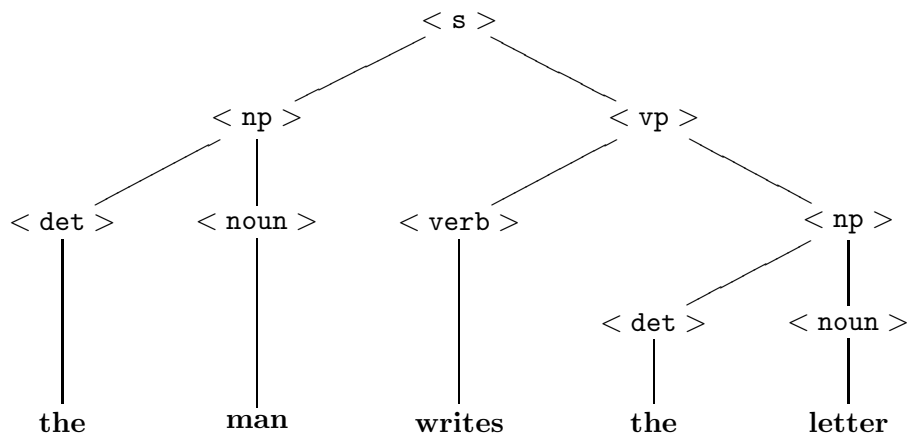


Figure 1.1: The structure of an English sentence.

Furthermore, a lexical substitution (according to some dictionary) is performed as a last step, in which the grammatical items  $\langle \text{determiner} \rangle$ ,  $\langle \text{noun} \rangle$  and  $\langle \text{verb} \rangle$  are replaced with the lexical items in boldface letters. Clearly, also other substitutions are possible, yielding other sentences with the same grammatical structure. For example, one might take the sentences

the cat eats the mouse

or

the letter writes the man

which explain clearly that grammatical correctness and structure only refer to the syntax of the sentences.

Similarly, as an example for the definition of the syntax of a programming language, rules for arithmetic expressions  $\langle \text{exp} \rangle$  are given. A part of those rules might look like this, see [3, Section 1.1.2]:

(r1)  $\langle \text{exp} \rangle ::= \langle \text{exp} \rangle + \langle \text{term} \rangle$

(r2)  $\langle \text{exp} \rangle ::= \langle \text{term} \rangle$

<sup>2</sup>As syntax trees are not formally treated in the present thesis, we refrain from defining the notion. For definitions we refer to the literature, for example, see [3] or [63].

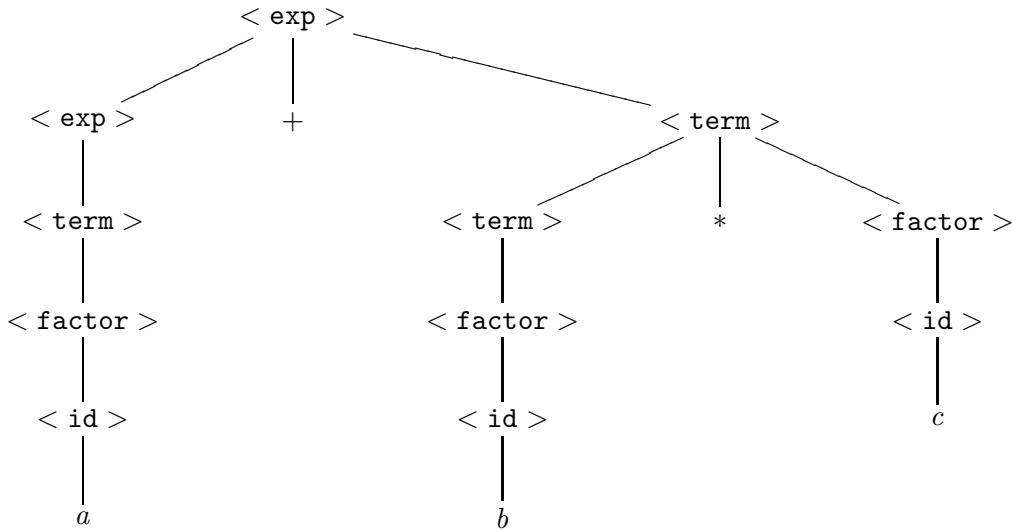


Figure 1.2: The structure of an arithmetic expression.

- (r3)  $\langle \text{term} \rangle ::= \langle \text{term} \rangle * \langle \text{factor} \rangle$   
 (r4)  $\langle \text{term} \rangle ::= \langle \text{factor} \rangle$   
 (r5)  $\langle \text{factor} \rangle ::= \langle \text{id} \rangle$

Then, for instance, the expression  $a + b * c$  has the syntax tree presented in Figure 1.2.

Again, there are several parses for the expression. If, for example, always the left-most grammatical item is selected to be replaced prior to any other item, then the parse  $r1r2r4r5r3r4r5r5$  is found.

## 1.2 Basic Definitions and Notation

In this section we introduce the notation used in the sequel and review notions from language and parsing theory that are a foundation for this thesis. All results presented in this section are preliminary and can be found in standard reference books. We refer to [63, 97, 99, 117] for a general background on formal languages and automata and to [3, 4, 104] for information on the theory of parsing.

Basic concepts from discrete mathematics (such as sets, relations, graphs, trees, etc.) are assumed to be known. By  $\mathbb{N}$  we denote the set of positive integers; then  $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ . For a singleton set  $\{s\}$  we write just  $s$  unless there is the risk of a confusion. As usual,  $\cup$ ,  $\cap$ , and  $\setminus$  are the operator symbols for set union, intersection, and set difference, respectively. Set inclusion and strict set inclusion are denoted by  $\subseteq$  and  $\subset$ , respectively. Let  $M$  be a set. Then,  $|M|$  is the cardinality of  $M$  and  $2^M$  is the set of all finite subsets of  $M$ .

### 1.2.1 Languages, Language Operations, Families of Languages

An alphabet is a finite and non-empty set. The elements of an alphabet are called symbols. Let  $V$  be an alphabet. A word over  $V$  is a finite concatenation of symbols taken from  $V$ . By  $V^*$  we denote the set of all words over  $V$  including the empty word  $\lambda$ . Then  $V^+ = V^* \setminus \{\lambda\}$

is the set of all non-empty words over  $V$ . For  $w \in V^*$ ,  $|w|$  is the length of  $w$ . The set of elements of  $V^*$  with length at most  $k$ , for some  $k \in \mathbb{N}$ , is denoted by  $V^{\leq k}$ . For  $U \subseteq V$ , the notation  $|w|_U$  is used for the number of occurrences of symbols from  $U$  in the word  $w$ , whereas  $\text{symb}(x) = \{a \in V \mid |x|_a > 0\}$  is the set of all symbols occurring in  $w$ . A *language* over  $V$  is a subset of  $V^*$ . A language is  $\lambda$ -free if it does not contain the empty word. For languages  $L$  and  $L'$ , their concatenation  $L \cdot L'$  is the set  $\{ww' \mid w \in L, w' \in L'\}$ . Then,  $L^0 = \{\lambda\}$ ,  $L^i = L \cdot L^{i-1}$ , for  $i \geq 1$ , and  $L^* = \bigcup_{i \geq 0} L^i$ . The language  $L^*$  is called the *Kleene closure* of  $L$ . For a language  $L$  and a word  $w$ ,  $d_w^l(L) = \{u \mid u \in V^*, wu \in L\}$  and  $d_w^r(L) = \{u \mid u \in V^*, uw \in L\}$  are the left and right derivatives of  $L$  by  $w$ , respectively. The *reversal* of a word over alphabet  $V$  is recursively defined by  $\lambda^R = \lambda$  and  $(va)^R = av^R$ , for  $v \in V^*$  and  $a \in V$ . Then, for  $L \subseteq V^*$ , the reversal is  $L^R = \{w^R \mid w \in L\}$ .

Consider words  $u, v, w \in V^*$  such that  $w = uv$ . Then  $u$  is a *prefix* of  $w$ , and  $v$  is a *suffix* of  $w$ . If  $w = v_1uv_2$  for some  $v_1, v_2 \in V^*$ , then  $u$  is a subword of  $w$ . Let  $\text{Pref}(w)$ ,  $\text{Suf}(w)$  and  $\text{Sub}(w)$  denote the set of prefixes, suffixes and subwords, respectively, of  $w$ . Furthermore, by  $\text{pref}_k(w)$ ,  $w \in V^*$ , we denote the prefix of length  $k$  of  $w$  if  $|w| \geq k$  or, otherwise, the string  $w$  itself. For a language  $L$ , let  $\text{pref}_k(L) = \{\text{pref}_k(w) \mid w \in L\}$ .

Let  $U$  and  $V$  be two alphabets, not necessarily different. Furthermore, let  $s : V \rightarrow 2^{U^*}$  be a mapping. The extension of  $s$  to  $V^*$  defined by  $s(\lambda) = \{\lambda\}$  and  $s(vw) = s(v) \cdot s(w)$ , for  $v, w \in V^*$ , is referred to as *substitution from  $V$  into  $U$* . If, for all  $a \in V$ , the set  $s(a)$  is finite, then  $s$  is a *finite substitution*. If, for all  $a \in V$ ,  $s(a)$  is a singleton set, then  $s$  is a *homomorphism*; if additionally  $s(a) \neq \lambda$ , for all  $a \in V$ , then the homomorphism is  $\lambda$ -free. For any substitution  $s$  from  $V$  into  $U$  and a language  $L \subseteq V^*$ ,  $s(L) = \bigcup_{w \in L} s(w)$ . For a homomorphism  $h : V^* \rightarrow U^*$  and  $L \subseteq U^*$ , the *inverse homomorphism of  $L$*  is defined by  $h^{-1}(L) = \{w \in V^* \mid h(w) \in L\}$ . Thus, for a word  $u \in U^*$ ,  $h^{-1}(u) = \{w \in V^* \mid h(w) = u\}$ .

A *generalized sequential machine* (gsm, for short) is a sextuple  $M = (Q, V, U, q_0, \tau, F)$ , where  $Q$  is a finite set of states,  $V$  and  $U$  are the alphabets of input and output symbols, respectively,  $q_0 \in Q$  is the initial state,  $F \subseteq Q$  is the set of final states, and  $\tau$  is the transition function,  $\tau : Q \times V \rightarrow 2^{(Q \times U^*)}$ .  $M$  is  $\lambda$ -free if  $\tau : Q \times V \rightarrow 2^{(Q \times U^+)}$ . Let  $w \in V^*$ . The *gsm-mapping* of  $w$  induced by  $M$  is defined in the following way. Let  $w = a_1a_2 \dots a_n$  with  $a_j \in V$ ,  $1 \leq j \leq n$ . Then,

$$g_M(w) = \{u_1u_2 \dots u_n \mid \text{there are states } q_{i_0}, q_{i_1}, \dots, q_{i_n} \text{ with} \\ q_{i_0} = q_0, q_{i_n} \in F \text{ and } (q_{i_j}, u_j) \in \tau(q_{i_{j-1}}, a_j) \text{ for } 1 \leq j \leq n\}.$$

The gsm-mapping  $g_M$  is extended to languages over  $V$  as usual by  $g_M(L) = \bigcup_{w \in L} g_M(w)$ . It is  $\lambda$ -free if  $M$  is  $\lambda$ -free.

Following [63], a *family of languages* is a set of languages containing at least one non-empty language. Let  $\mathcal{L}$  be a family of languages and  $o$  be a  $k$ -ary language operation. The family  $\mathcal{L}$  is *closed* under the language operation  $o$  if  $L_1, L_2, \dots, L_k \in \mathcal{L}$  implies  $o(L_1, L_2, \dots, L_k) \in \mathcal{L}$ . A family of languages is referred to as *Abstract Family of Languages (AFL)* if it is closed under union, concatenation, Kleene closure,  $\lambda$ -free homomorphisms, inverse homomorphisms and intersection with regular languages (with respect to *any*  $\lambda$ -free homomorphism, inverse homomorphism and regular language, respectively), where a language over some alphabet  $V$  is *regular* if it can be obtained by a finite number of applications of the operations union, concatenation and Kleene closure to the elements of  $V \cup \{\lambda\}$ . An AFL is *full* if it is closed with respect to arbitrary homomorphisms. Any full AFL is closed with respect to gsm-mappings, and any AFL is closed with respect to  $\lambda$ -free gsm-mappings and left and right derivatives.

A finite language can be defined by enumerating its elements. Apart from that there are three principal ways of defining languages with the help of a finite specification:

- definition by construction (for instance, in a recursive manner or algebraically as in the case of regular languages),
- generation,
- acceptance.

The latter two ways of defining languages are treated in the next section.

We term two devices describing languages equivalent if the two described languages are equal. Two languages  $L_1$  and  $L_2$  are considered to be equal if and only if  $L_1 \setminus \{\lambda\} = L_2 \setminus \{\lambda\}$ . We simply write  $L_1 = L_2$  in this case.

### 1.2.2 Automata, Grammars and the Chomsky Hierarchy

Automata are devices which can be used for language *acceptance*. For example, *finite state automata* are, in principle, generalized sequential machines without output. A finite state automaton accepts an input word if it can reach an accepting (final) state by reading the input completely. More precisely, a nondeterministic finite state automaton (NFA, for short) is a quintuple  $A = (Q, V, q_0, \delta, F)$ , where  $Q$  is a finite set of states,  $V$  is the alphabet of input symbols,  $q_0 \in Q$  is the initial state,  $F \subseteq Q$  is the set of accepting states, and  $\delta$  is the transition function,  $\delta : Q \times V \rightarrow 2^Q$ . The transition function is extended to words over  $V$  by  $\delta(q, \lambda) = \{q\}$  and  $\delta(q, va) = \bigcup_{p \in \delta(q, v)} \delta(p, a)$ , for  $q \in Q$ ,  $v \in V^*$  and  $a \in V$ . The language accepted by  $A$  is the set

$$L(A) = \{w \in V^* \mid \delta(q_0, w) \cap F \neq \emptyset\}.$$

A finite state automaton is *deterministic* (a DFA) if, for all  $q \in Q$  and  $a \in V$ ,  $|\delta(q, a)| = 1$ .

The capability of finite state automata can be extended in several ways, for example, by adding further resources. In *pushdown automata*, they are equipped with a pushdown store as additional memory which is used, besides the finite set of states, in order to control the computation; for more details and formal definitions see, for example, [63]. If the automata have an unlimited read-write memory, then one is led to the concept of Turing machines which are computationally complete in the sense that any function which is viewed to be effectively computable (for example by computer programs) can also be computed by a Turing machine and *vice versa*, possibly after appropriate encodings of the input and the output. Formally, a *nondeterministic Turing machine* is defined to be a tuple  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ , where  $Q$  is the set of states,  $\Gamma$  is the alphabet of tape symbols,  $B \in \Gamma$  is the blank symbol,  $\Sigma \subseteq \Gamma \setminus \{B\}$  is the alphabet of input symbols,  $\delta$  is the transition function mapping from  $Q \times \Gamma$  to  $2^{Q \times \Gamma \times \{L, R\}}$ ,  $q_0 \in Q$  is the initial state, and  $F \subseteq Q$  is the set of final states. Similarly to the case of finite state automata, the Turing machine  $M$  is *deterministic* if  $|\delta(q, a)| = 1$  for all  $q \in Q$  and  $a \in \Gamma$ .

A configuration characterizing the global state of  $M$  is written as  $xqy$ , where  $q \in Q$  is the current state and  $xy \in \Gamma^*$  is the word currently stored in the read-write memory of  $M$ . The memory is thought to be an infinite tape, divided into cells, each of which containing a symbol from  $\Gamma$ . The cells to the left and the right of  $xy$  are filled with the blank symbol. Therefore, there is an infinite number of configurations characterizing one and the same global state,

since one can write as many as needed of those blank symbols, as a prefix of  $x$  and a suffix of  $y$ .

A move of  $M$  is defined as follows. Let  $a_1a_2 \dots a_{i-1}qa_ia_{i+1} \dots a_n$  be a configuration and  $(p, b, R) \in \delta(q, a_i)$ . Then,

$$a_1a_2 \dots a_{i-1}qa_ia_{i+1} \dots a_n \vdash_M a_1a_2 \dots a_{i-1}bpa_{i+1} \dots a_n.$$

If  $(p, b, L) \in \delta(q, a_i)$ , then

$$a_1a_2 \dots a_{i-1}qa_ia_{i+1} \dots a_n \vdash_M a_1a_2 \dots a_{i-2}pa_{i-1}ba_{i+1} \dots a_n.$$

The language accepted by  $M$  is the set

$$L(M) = \{ w \mid w \in \Sigma^*, q_0w \vdash_M^* xpy \text{ for some } p \in F \text{ and } x, y \in \Gamma^* \},$$

where  $\vdash_M^*$  denotes the reflexive and transitive closure of the relation  $\vdash_M$ .

For every nondeterministic Turing machine there is a deterministic one simulating it. This is important with respect to the following observation. From the point of view of computations, every Turing machine induces a function. If, for  $w \in \Sigma^*$ ,  $xy \in \Gamma \setminus \{B\}^*$ ,  $q_0w \vdash_M^* xpy$  holds for some  $p \in F$ , then  $xy$  is the value of the induced function with argument  $w$ . Thus, a language is accepted by a Turing machine if and only if it is the domain of a function induced by a Turing machine (a computable function). Equivalently, it is the range of a computable function. Such language is said to be *recursively enumerable*. A language  $L$  over  $\Sigma$  is *recursive* if there is a Turing machine with input alphabet  $\Sigma$  which halts on each input from  $\Sigma^*$  answering the question whether or not  $w \in L$ . The families of all recursively enumerable and all recursive languages are denoted by  $\mathcal{L}(\text{RE})$  and  $\mathcal{L}(\text{REC})$ , respectively.

Formal grammars are the traditional mean in order to *generate* languages. A *phrase structure grammar* is a quadruple  $G = (V_N, V_T, P, S)$ , where  $V_N$  and  $V_T$  are disjoint alphabets,  $P$  is a finite subset of  $(V_N \cup V_T)^*V_N(V_N \cup V_T)^* \times (V_N \cup V_T)^*$ , and  $S \in V_N$ . The sets  $V_N$  and  $V_T$  are the alphabets of nonterminal symbols and terminal symbols, respectively,  $P$  is the set of productions (or rewriting rules), and  $S$  is the start symbol (or axiom) of  $G$ . The set  $V_G = V_N \cup V_T$  is the total alphabet of  $G$ . In what follows, we write  $\alpha \rightarrow \beta$  for  $(\alpha, \beta) \in P$ .

For words  $\gamma$  and  $\gamma'$  over  $V_G$ ,  $\gamma$  directly derives  $\gamma'$  in  $G$  if and only if there are words  $\delta_1, \delta_2 \in V_G^*$  and a production  $\alpha \rightarrow \beta \in P$  such that  $\gamma = \delta_1\alpha\delta_2$  and  $\gamma' = \delta_1\beta\delta_2$ . We write  $\gamma \Rightarrow_G \gamma'$  in this case. The subscript  $G$  is omitted when there is no risk of a confusion. A sequence of words  $\gamma_0, \gamma_1, \dots, \gamma_n$  is referred to as a *derivation* in  $G$  if  $\gamma_{i-1} \Rightarrow \gamma_i$ , for  $1 \leq i \leq n$ . We use the notation  $\gamma_0 \xRightarrow{n} \gamma_n$  expressing the fact that the derivation consists of exactly  $n$  steps. Moreover, we write  $\gamma \xRightarrow{*} \gamma'$  if  $\gamma \xRightarrow{n} \gamma'$  for some  $n \geq 0$ . Thus,  $\xRightarrow{*}$  denotes the reflexive and transitive closure of the relation  $\Rightarrow$ . This notation is also used for the yield relations of other grammar types, defined in the sequel.

A word  $\gamma \in V_G^*$  with  $S \xRightarrow{*} \gamma$  is said to be a *sentential form* of  $G$ . The *language*  $L(G)$  generated by  $G$  is the set of all sentential forms of  $G$  which are contained in  $V_T^*$ , that is,

$$L(G) = \{ w \mid w \in V_T^* \text{ and } S \xRightarrow{*} w \}.$$

One distinguishes four basic types of phrase structure grammars. A phrase structure grammar  $G = (V_N, V_T, P, S)$  is called

- (i) *type-0 grammar* if there is no restriction on the set of production rules (that is, any phrase structure grammar is a type-0 grammar and *vice versa*),

- (ii) *type-1 grammar* if, for each production  $\alpha \rightarrow \beta$  in  $P$ , there are words  $\eta_1, \eta_2$  over  $V_G$ ,  $A \in V_N$  and  $\gamma \in V_G^+$  such that  $\alpha = \eta_1 A \eta_2$  and  $\beta = \eta_1 \gamma \eta_2$ ,
- (iii) *type-2 grammar* if, for each production  $\alpha \rightarrow \beta$  in  $P$ ,  $\alpha \in V_N$ ,
- (iv) *type-3 grammar* if, for each production  $\alpha \rightarrow \beta$  in  $P$ ,  $\alpha \in V_N$  and  $\beta \in V_T^* V_N \cup V_T^*$ .

That is, any type-3 grammar is a type-2 grammar, where it is additionally required that on the right-hand sides of the rules at most one nonterminal symbol occurs and if so, then at the rightmost position. Therefore, type-3 grammars are also referred to as *right-linear grammars*. One could also define left-linear grammars where, for each production  $\alpha \rightarrow \beta$  in  $P$ ,  $\alpha \in V_N$  and  $\beta \in V_N V_T^* \cup V_T^*$  is required. Right- and left linear grammars are able to generate exactly the same languages, namely the regular languages. More precisely, for any regular language there is both a right-linear and a left-linear grammar generating it and, conversely, any right-linear or left-linear grammar generates a regular language. Therefore, type-3 grammars are also called *regular grammars*. Type-2 and type-1 grammars are referred to as *context-free* and *context-sensitive grammars*, respectively.

Moreover, one considers the following types of phrase structure grammars. A context-free grammar  $G = (V_N, V_T, P, S)$  is *linear* if  $\alpha \rightarrow \beta \in P$  implies  $\beta \in V_T^* V_N V_T^* \cup V_T^*$ . A phrase structure grammar is *monotone* if, for each production  $\alpha \rightarrow \beta$  in  $P$ ,  $|\alpha| \leq |\beta|$ .

A language  $L$  is said to be of type- $i$ ,  $0 \leq i \leq 3$ , right-linear, left-linear, regular, linear, context-free, context-sensitive or monotone if there is a type- $i$  grammar,  $0 \leq i \leq 3$ , a right-linear, left-linear, regular, linear, context-free, context-sensitive or monotone grammar, respectively, generating  $L$ . In order to compare the generative powers of the various types of grammars, the families of languages are considered which are defined by the classes of all grammars of one and the same type. By  $\mathcal{L}(\text{REG})$ ,  $\mathcal{L}(\text{LIN})$ ,  $\mathcal{L}(\text{CF})$ ,  $\mathcal{L}(\text{CS})$  and  $\mathcal{L}(\text{MON})$  the families of all regular (or, equivalently, right-linear or left-linear), linear, context-free, context-sensitive and monotone languages are denoted, respectively. Finally, the family of all type-0 languages is  $\mathcal{L}(\text{RE})$ , since type-0 grammars generate precisely all recursively enumerable languages.

The other families of type- $i$  languages have a characterization in terms of automata, either. The family of languages accepted by DFAs (or, equivalently, by NFAs) is the family of regular languages, whereas the families of context-free and context-sensitive languages are characterized by the classes of nondeterministic pushdown automata<sup>3</sup> and linear-bounded automata. Here a *linear-bounded automaton* (LBA, for short) is a non-deterministic Turing machine in which the number of cells that can be used during a computation on an input word  $w$  is  $O(|w|)$ . That is, given an input word  $w$ , the number of cells available on the work tape is bounded by a function which is linear in the length  $|w|$  of the input.<sup>4</sup> A linear-bounded automaton is said to be *deterministic* if and only if the underlying Turing machine is deterministic. It is an open question whether or not deterministic linear-bounded automata are as powerful as non-deterministic LBAs. This question is referred to as the LBA problem [69].

A fundamental result on context-free grammars is that erasing productions can effectively be eliminated without affecting the generated language. Generally, a grammar is  $\lambda$ -free if it

<sup>3</sup>It is known that, in contrast to finite state automata and Turing machines, deterministic pushdown automata are strictly less powerful than their nondeterministic variants.

<sup>4</sup>Formally, for two functions  $f : \mathbb{N}_0 \rightarrow \mathbb{N}_0$  and  $g : \mathbb{N}_0 \rightarrow \mathbb{N}_0$ ,  $f$  is  $O(g)$  if there are positive integers  $n_0$  and  $c$  such that, for all  $n \geq n_0$ ,  $f(n) \leq cg(n)$ .

contains no  $\lambda$ -rule, that is, each of its rules  $\alpha \rightarrow \beta$  satisfies  $\beta \in V_G^+$ . As any production of a  $\lambda$ -free context-free grammar is a context-sensitive one (with  $\eta_1 = \eta_2 = \lambda$ ), it follows that any context-free language is context-sensitive. Therefore, any type- $i$  language is a type- $(i-1)$  language, for  $1 \leq i \leq 3$ . Furthermore, there is a type- $(i-1)$  language which is no type- $i$  language,  $1 \leq i \leq 3$ . These results together establish the *Chomsky hierarchy*:

$$\mathcal{L}(\text{REG}) \subset \mathcal{L}(\text{CF}) \subset \mathcal{L}(\text{CS}) \subset \mathcal{L}(\text{RE}).$$

Further results extend this hierarchy as follows:

$$\mathcal{L}(\text{FIN}) \subset \mathcal{L}(\text{REG}) \subset \mathcal{L}(\text{LIN}) \subset \mathcal{L}(\text{CF}) \subset \mathcal{L}(\text{CS}) = \mathcal{L}(\text{MON}) \subset \mathcal{L}(\text{REC}) \subset \mathcal{L}(\text{RE}),$$

where  $\mathcal{L}(\text{FIN})$  is the family of all finite languages.

The result that, for any context-free grammar there is an equivalent  $\lambda$ -free context-free one can be strengthened by the following normal form result: A context-free grammar  $G = (V_N, V_T, P, S)$  is in *Chomsky normal form* if each production in  $P$  is of the form  $A \rightarrow BC$  or  $A \rightarrow a$ ,  $A, B, C \in V_N$  and  $a \in V_T$ .<sup>5</sup> For any context-free grammar, one can effectively construct an equivalent grammar in Chomsky normal form. A similar normal form result is known for context-sensitive grammars. For any context-sensitive grammar, there is effectively an equivalent grammar in *Kuroda normal form*, where all rules are of one of the following forms:  $A \rightarrow BC$ ,  $AB \rightarrow CD$ , or  $A \rightarrow a$ , where  $A, B, C, D \in V_N$ ,  $a \in V_T$ .

### 1.2.3 Basic Elements of the Theory of Parsing

A parsing algorithm is an effective procedure which, given a grammar  $G$  and a terminal word  $w$  as input, decides whether  $w$  can be generated by  $G$ , and if so, reconstructs (at least) one derivation. Parsing algorithms which are known to work for all context-free grammars, such as the Cocke-Younger-Kasami algorithm or the parsing method by Earley, see [3], need to take  $O(n^3)$  steps, when the input is of length  $n$ . The algorithms can partially be improved due to techniques for fast multiplication of matrices, but they keep a time complexity which is strictly worse than quadratic in the input length. In order to obtain more efficient parsing algorithms, one has to restrict context-free grammars so that the derivations become “more deterministic”. In a certain sense, context-free grammars possess a threefold nondeterminism. Let  $G = (V_N, V_T, P, S)$  be a context-free grammar. In a derivation step according to  $G$  in which a sentential form  $\gamma$  shall be rewritten, one may

1. select a nonterminal symbol  $A \in V_N$  which shall be replaced;
2. select an occurrence of  $A$  in  $\gamma$  to be replaced;
3. select a production of the form  $A \rightarrow \alpha$ , that is, the right-hand side  $\alpha$  with which  $A$  shall be rewritten.

Some useful ways in which this nondeterminism can be restricted, are explained in the following.

---

<sup>5</sup>In the literature, one frequently finds definitions of the Chomsky normal form in which additionally a rule of the form  $S \rightarrow \lambda$  is allowed under the condition that the symbol  $S$  does not occur on the right-hand side of any production in  $P$ . The same addition is found in the definitions of context-sensitive grammars. As this rule  $S \rightarrow \lambda$  is only needed in order to generate the empty word, we prefer to omit it in the definitions, but allow to consider two languages to be equal if they differ from each other at most by the empty word.

Consider a derivation  $\gamma_1 \Rightarrow \gamma_2 \Rightarrow \dots \Rightarrow \gamma_n$  in  $G$ . If in any  $\gamma_i$ , the leftmost occurrence of a nonterminal is replaced, then this derivation is called *leftmost derivation* in  $G$ . That is, for  $1 \leq i < n$ , if  $\gamma_i = u_i A_i \beta_i$  with  $u_i \in V_T^*$ ,  $A_i \in V_N$  and  $\beta_i \in V_G^*$ , then  $\gamma_{i+1} = u_i \alpha_i \beta_i$  has to hold, for some  $\alpha_i$  with  $A_i \rightarrow \alpha_i \in P$ .<sup>6</sup> We write  $\gamma_i \xrightarrow{\text{lm}}^* \gamma_{i+1}$  in this case and  $\xrightarrow{\text{lm}}^*$  for the reflexive and transitive closure of  $\xrightarrow{\text{lm}}$ . On the one hand, for any context-free grammar  $G$ , the language generated only by leftmost derivations is equal to  $L(G)$ , where the order in which the nonterminal symbols are replaced is arbitrary. On the other hand, there are *ambiguous* context-free grammars in which some words have several different leftmost derivations. Even worse, there are context-free languages which are inherently ambiguous, that is, any context-free grammar generating it is ambiguous.

One subclass of context-free grammars which are unambiguous and for which parsing algorithms exist that take  $O(n)$  steps, when  $n$  is the length of the input, is the class of context-free LL( $k$ ) grammars. A context-free grammar  $G = (V_N, V_T, P, S)$  satisfies the LL( $k$ ) property, for some positive integer  $k$ , if whenever there are two leftmost derivations

$$(1) S \xrightarrow{\text{lm}}^* uA\beta \xrightarrow{\text{lm}}^* w\alpha\beta \xrightarrow{\text{lm}}^* uv \text{ and}$$

$$(2) S \xrightarrow{\text{lm}}^* uA\beta \xrightarrow{\text{lm}}^* w\alpha'\beta \xrightarrow{\text{lm}}^* wv'$$

such that  $\text{pref}_k(v) = \text{pref}_k(v')$ , it follows that  $\alpha = \alpha'$ . Intuitively, given (1) a sentential form  $uA\beta$  of a leftmost derivation in  $G$  generating the terminal word  $uv$ , for some word  $v$ , and (2) the first  $k$  symbols of  $v$  (if they exist), there is only one production in  $P$  which can be applied to  $uA\beta$  in the next leftmost derivation step. For each  $k \geq 1$ , every LL( $k$ ) grammar is LL( $k+1$ ) and there is a language for which an LL( $k+1$ ) grammar but no LL( $k$ ) grammar exists, see [94].

Let  $k \geq 1$  and  $G = (V_N, V_T, P, S)$  be a context-free LL( $k$ ) grammar. For an effective construction of a parser for  $G$  one needs to know the sets of all words in  $L^{\leq k}$  which are prefixes of those terminal words that can be derived from single nonterminal symbols or may appear immediately to the right of them. More generally, for any  $\alpha \in (V_N \cup V_T)^*$ , let

$$\text{FIRST}_k^G(\beta) = \text{pref}_k(L(G, \beta)),$$

where  $L(G, \beta)$  denotes the set of terminal words that can be derived from  $\beta$ ,

$$L(G, \beta) = \{w \in V_T^* \mid \beta \xrightarrow{G}^* w\}.$$

Furthermore,

$$\text{FOLLOW}_k^G(\beta) = \{w \in V_T^* \mid \text{there is } \alpha, \gamma \text{ with } S \xrightarrow{G}^* \alpha\beta\gamma \text{ and } w \in \text{FIRST}_k(\gamma)\}.$$

The symbol  $G$  is omitted if no confusion may arise. For an effective construction of these sets and of a parser for context-free LL( $k$ ) grammars, see for example [3]. It is also pointed out there that the construction of the parser becomes particularly simple if the leftmost nonterminal symbol together with the first  $k$  terminal symbols to be derived next (namely  $A$  and  $\text{pref}_k(v)$  in the definition of the LL( $k$ ) property) are sufficient to determine the production which has to be used rewriting the leftmost nonterminal. Systems with this property are said

<sup>6</sup>Alternatively, one can define rightmost derivations or other manners in which the sentential forms have to be rewritten. Those are not treated in the present thesis.



to be *strong* LL( $k$ ). Formally, a context-free grammar  $G = (V_N, V_T, P, S)$  is strong LL( $k$ ) if the following statement holds:

$$\begin{aligned} &\text{If } A \rightarrow \beta \text{ and } A \rightarrow \beta' \text{ are distinct productions in } P, \text{ then} \\ &\text{FIRST}_k(\beta\text{FOLLOW}_k(A)) \cap \text{FIRST}_k(\beta'\text{FOLLOW}_k(A)) = \emptyset. \end{aligned}$$

Every context-free LL(1) grammar is strong LL(1).

### 1.3 Insufficiency of Context-Freeness

As explained in Section 1.1, there are syntactical issues of programming languages which are not checked during the phase of syntactical analysis. The reason why such checks have to be shifted to other phases is that there are some aspects in the syntax of many high-level programming languages which cannot be expressed with the help of context-free grammars (thus, in BNF). One typical example is that identifiers are required to be declared prior to their use. In [54], Floyd has provided a formal argument that this aspect cannot be captured in terms of context-free grammars for ALGOL 60, see also [41, Section 0.4]. We present this argument using the example of the language Java. Let us consider a class of the form

```
class NonCF {
 void method() {
 int x;
 y = 1;
 }
}
```

A source code like this is correct if and only if  $x$  and  $y$  are identical identifiers. Let  $R$  be the regular set of all Java classes which are of the form as above with arbitrary identifiers  $x, y \in \{a, b\}^*$ , and let JAVA be the set of all correct source codes in Java. Furthermore, let  $g$  be the gsm which erases all symbols different from those in  $x$  and  $y$ . Then

$$g(\text{JAVA} \cap R) = \{xx \mid x \in \{a, b\}^+\}$$

is not context-free. Since the family of context-free languages is a full AFL, thus closed with respect to gsm-mappings and intersections with regular sets, the language JAVA is not context-free.

As already Floyd pointed out in [54], it seems that this argument can be adapted to any other “reasonable language in which all variables must be declared”.

Together with this example, Dassow and Păun presented seven circumstances where context-free languages turn out to be insufficient, emerging in the field of programming languages, the language of logic, graph theory, developmental biology, economic modeling, folklore, and natural languages. In the remainder of this section, this list of examples is supplemented by two items.

#### Molecular Genetics

DNA molecules consist of double strands of nucleotides wound to a helix. Due to the Watson-Crick complementarity of the involved nucleotides and a particular molecule structure at the ends of the strands (denoted by 3' and 5'), a DNA molecule can be represented as a word over

a four-letter alphabet  $\{A, C, G, T\}$ , each letter representing one of the nucleotides. According to the Watson-Crick complementarity,  $A$  is in relation with  $T$  and  $C$  is in relation with  $G$ . For more details, see [61, 90].

The DNA, together with proteins, is packaged to chromosomes. At the end of a chromosome, particular DNA sequences, called *telomeres*, appear which do not contain genetic information and protect the chromosomes from degeneration and fusion with other chromosomes. During the process of DNA replication, several enzymes cause the double helix to unwind and partially split into two single strands. Due to the complementarity of the nucleotides, each of the single strands serves as template for the formation of double strands, again. This duplication starts not at the very ends of the strands. Therefore, a piece of the telomeres is lost during each DNA replication, causing cell aging. That is also why the life of cells is restricted. Another enzyme, namely telomerase, can prolongate the telomeres again. Telomerase consists of a protein and an RNA part; RNA molecules are similar to DNA single strands, but one of the nucleotides is replaced. Therefore, in the alphabet for RNA strings, the letter  $U$  is used instead of  $T$ .

A complex secondary structure (folding) of the telomerase RNA, so-called *pseudoknots*, is critical for its biological activity [26]. Pseudoknots consist of at least two hairpin loops and contain *crossing sequences* as it is seen in the schema given in Figure 1.3, see [83].

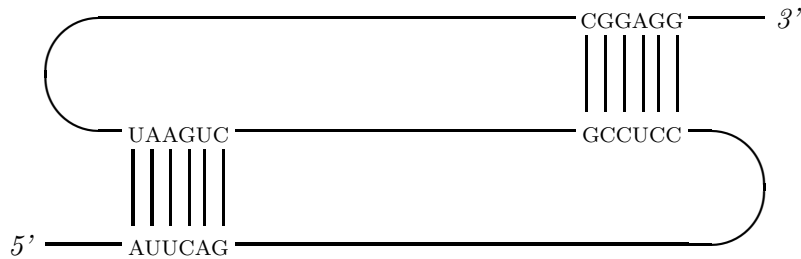


Figure 1.3: Crossing sequences in pseudoknots—a schematic representation.

There are hydrogen bonds between nucleotides appearing in the single strand, obeying the Watson-Crick complementarity. These bonds are represented as vertical lines in Figure 1.3. If we label the blocks, in this example AUUCAG, CCUCCG, CUGAAU, and CCGAGG, read in the  $5'$ - $3'$  direction, by  $i, j, i', j'$ , the hydrogen bonds appear between  $i$  and  $i'$  as well as  $j$  and  $j'$ . Since  $i < j < i' < j'$ , where  $<$  denotes the “occurs-left-to” relation, the pairwise bonded blocks occur in a crossing sequence. That is, the telomerase RNA strand is of the form  $w_1 w_2 w'_1 w'_2$  where  $w_i$  is linked via the Watson-Crick complementarity with  $w'_i$ ,  $1 \leq i \leq 2$ , at least if symbols are ignored which are not relevant in this respect. Then the set of those structures is not context-free since (if the bonding is encoded) it can be mapped to the non-context-free language  $\{a^m b^n c^m d^n \mid m, n \geq 1\}$  with the help of a gsm.

## Natural Languages

In order to prove the non-context-freeness of some natural language it is essential to find cross-serial dependencies of arbitrary large size in this language. Many of those arguments given in the literature turned out to be contestable because they exploit features which are

rather anomalous from a semantic or a pragmatic point of view than ungrammatical, see [92]. Nevertheless, evidence has been achieved for the non-context-freeness of some particular languages such as Swiss-German, keeping the arguments on a pure syntactical level. The reader may confer [103] or [84, Chapter 18] and [35] for another example.

Consider the following sentences in Swiss-German with their translations into English (first word-by-word, then into grammatical English sentences):

- **Jan säit das mer em Hans es huus hälfed aastriiche.**  
John said that we Hans the house helped paint.  
*John said that we helped Hans paint the house.*
- **Jan säit das mer d'chind em Hans es huus lönd hälfe aastriiche.**  
John said that we the children Hans the house let help paint.  
*John said that we let the children help Hans paint the house.*
- **Jan säit das mer (d'chind)<sup>i</sup> (em Hans)<sup>j</sup> es huus haend wele (laa)<sup>i</sup> (hälfe)<sup>j</sup> aastriiche.**  
John said that we (the children)<sup>i</sup> (Hans)<sup>j</sup> the house have wanted to (let)<sup>i</sup> (help)<sup>j</sup> paint.  
*John said that we wanted to let Mary help Hans, let Frank help Jessica, let Chris help Lucy, and let Vanessa help René paint the house.*

In the first sentence, the verbs (*hälfed* and *aastriiche*) occur in the same order as the objects (*em Hans* and *es huus*) they refer to. Thus, it demonstrates that crossing sequences may be present in Swiss-German sentences. Next, the sentence is modified in a way such that the last sentence shows how it can be extended to arbitrary length in a reasonable way; its latter translation provides an example sentence with  $i = j = 4$ . If a gsm is applied to the last sentence is Swiss-German, which maps

- verbs requiring dative case (*hälfe*) to  $a$ ,
- verbs requiring accusative case (*laa*) to  $b$ ,
- dative case objects (*Hans*) to  $c$ ,
- accusative case objects (*d'chind*) to  $d$ , and
- everything else to  $\lambda$ ,

then the resulting word is  $a^i b^j c^i d^j$ . The gsm-mapping of the intersection of Swiss-German with the regular set

Jan säit das mer (d'chind)\* (em Hans)\* es huus haend wele (laa)\* (hälfe)\* aastriiche.

yields the non-context-free language  $\{ a^m b^n c^m d^n \mid m, n \geq 1 \}$ , again. Hence, Swiss-German is not context-free.

### Non-Context-Free Mechanisms

Alas, the next more powerful type of Chomsky grammars, namely the context-sensitive grammars, are also not used in most applications because they are too complex. For example, the fixed membership problem (thus, the parsing problem) is **PSPACE**-complete and many other relevant decision problems are proved to be undecidable<sup>7</sup> for context-sensitive grammars.

Therefore, a series of grammar formalisms has been introduced which are able to cover all the desired non-context-free aspects but aim to maintaining the nice properties of context-free grammars. The fact that there are non-context-free aspects in the syntax of natural languages led to the concept of *mildly context-sensitive grammars* [65] providing some lower and upper bound conditions for families of languages which may be useful in linguistics, see also [9]. A grammar formalism is said to be mildly context-sensitive if the family of its languages

1. contains (besides the context-free languages<sup>8</sup>)

$$\begin{aligned} L_1 &= \{ a^n b^n c^n \mid n \geq 1 \}, \\ L_2 &= \{ w c w \mid w \in \{a, b\}^* \}, \\ L_3 &= \{ a^m b^n c^m d^n \mid m, n \geq 1 \}, \end{aligned}$$

2. contains only languages which can be parsed in polynomial time, and
3. contains only semilinear languages.<sup>9</sup>

Besides tree-adjoining grammars introduced in [66] and other mildly context-sensitive grammars [110], the three most important sources of such language describing devices are grammars with controlled derivations, grammars with parallel derivations (mainly Lindenmayer systems), and grammar systems; a survey about those mechanisms is presented in the next chapter.

---

<sup>7</sup>For definitions of the notions from complexity and recursion theory, the reader is referred, for example, to [63].

<sup>8</sup>This requirement to cover all context-free languages has been weakened in more recent articles.

<sup>9</sup>Originally, in [65] Joshi required the slightly weaker property that, for all infinite languages  $L$  which can be generated, there is a constant  $C$  such that for any word  $w \in L$  there is another word  $w' \in L$  with  $0 < |w'| - |w| < C$ .

## Chapter 2

# Grammar Formalisms for Describing Non-Context-Free Languages

Grammar formalisms suitable for the description of formal languages which are not context-free have been considered since the beginning of formal language theory. One of the most famous class of those grammars might be the context-sensitive one, as it belongs to the Chomsky hierarchy. Due to its mathematical and algorithmic inadequacy, a bunch of other grammar formalisms which are more powerful than context-free grammars have been introduced and investigated. The underlying concepts can be divided into the following categories<sup>1</sup>, where we restrict ourselves to grammars with derivations in which symbols or substrings are substituted according to productions.

### 1. Grammars with controlled derivations.

Basically, a grammar with controlled derivations is an ordinary Chomsky grammar, say a context-free grammar, with its usual strongly nondeterministic derivation process. Out of the rich diversity of possible derivations, the control selects some to be valid, and disregards all the others.

In general, the control adds to the power of the type of the underlying grammar since the generated language can potentially be restricted to words possessing some desired properties. Note that the control can also regard all possible derivations to be valid, thus resuming the power of the underlying system. Here, the focus is on *context-free* grammars with controlled derivations.

Examples of control mechanisms are

- (a) a *restriction of the order* in which productions can be applied such as prescribed sequences of productions (like in matrix grammars [1] or grammars with regular control [58]) or a mechanism prescribing which productions are allowed to be applied in the next derivation step, depending on the production which has just been used (like in programmed grammars [93]); another example are grammars controlled by a bicolored digraph [116].

---

<sup>1</sup>As a matter of course, also other classifications are possible and can be found in the literature.

- (b) a *context condition* requiring that certain symbols are present or absent in the sentential form (like in random context grammars [112] or ordered grammars [56]) or even at certain positions (like in conditional grammars [85]); note that context-sensitive grammars can be considered as belonging to this category, the underlying Chomsky grammar being context-free,
- (c) *additional symbols inside the sentential forms*, encoding information about the derivation yielding the sentential forms (like in indexed grammars [2]) or allowing different nonterminals to communicate with each other in a restricted way (like in synchronizing grammars [68], see also [20]).

For a comprehensive monograph about grammars with controlled derivations we refer to [41]; an extensive survey containing also more recent results is given in [44].

## 2. Parallel or partially parallel grammars.

Whereas in Chomsky grammars the derivation process is sequential, that is a single occurrence of a symbol or a substring of the sentential form is replaced according to a production, in parallel grammars all symbols of the sentential form are replaced simultaneously. Thus, a finite substitution is repeated by those mechanisms. The various classes of *Lindenmayer systems* (L systems, for short) have been introduced to model the biological development of lower organisms [62]. Subsequently a sophisticated mathematical theory of L systems has been established [96].

Grammars with *partial parallelism* rewrite the sentential forms in a manner which is intermediate between sequential and parallel replacement. For example, uniformly limited L systems replace (if possible) a fixed number of symbols per derivation step [115], limited L systems replace a fixed number of occurrences of any symbol of the alphabet [113, 114], and Indian parallel grammars replace all occurrences of exactly one symbol of the alphabet whereas the other symbols remain unchanged [105].

## 3. Grammar systems.

A grammar system consists of several Chomsky grammars, called components of the system, which jointly generate a common language. The components perform derivation steps sequentially or in a parallel way.

- (a) *Sequential grammar systems* work on a common sentential form in turns, according to some cooperation strategy. The possibly best known model is the concept of cooperating distributed grammar systems, where all components are context-free grammars.

Typical cooperation strategies rely on counting the number of derivation steps consecutively performed by any component. Another approach is based on the “competence” of the components. In the terminating mode, any component, once started, has to continue rewriting until it has no production left which is applicable to the sentential form. In the full competence mode, a component remains active until and unless one nonterminal appears in the sentential form which cannot be rewritten.

Cooperating distributed grammar systems have been introduced as a grammatical model of distributed problem solving, following the blackboard architecture from artificial intelligence [31], see also [32], after a forerunner paper pursued a similar

approach as a generalization of two-level substitution (van-Wijngaarden) grammars to a multi-level concept [75]. Furthermore, in [12] CD grammar systems are considered as sequential counterparts to tabled Lindenmayer systems with auxiliary symbols (ETOL systems), which consist of several production sets (the tables) as well (see [95]).

- (b) In *parallel grammar systems*, the components work simultaneously on individual sentential forms in a synchronized way, according to a universal clock. They are able to communicate with each other by sending their sentential forms upon request (or command) to other components. Then, the generated language consists of all words obtainable by a designated component in this way.

Those parallel communicating grammar systems have been introduced as a grammatical model of parallelism in a very broad sense [91] and were associated with the classroom model of problem solving [32]: the work in a real classroom or in similar environments, for instance in a research team, is organized by distributing the task to several workers which perform problem solving steps simultaneously but individually and which are allowed to exchange information about the state of the problem solving, possibly according to some restricted communication protocol. From another point of view, parallel communicating grammar systems are language generating models of parallel and distributed computation as it appears, for example, in computer nets. This approach has several advantages. So it allows to compare the power of distinct communication structures (parallel architectures) in a way in which other (computing) models have not been able to establish results (see [64]).

For standard references about grammar systems, including also further types such as eco-grammar systems or test tube systems, confer [32] and [43].

Several types of grammars combine some of the features listed above. Scattered context grammars, for example, rewrite the sentential forms in a partially parallel manner, simultaneously obeying context constraints, see [59] (or [78] for the unordered variant). Furthermore, one might consider systems of (partially) parallel grammars or adding control mechanisms to those grammars or grammar systems. One approach to mechanisms allowing combinations in a very general sense is given by the concept of *networks of language processors* [30].

## 2.1 Definitions

The formal definitions of the grammars and systems which are in the focus of the present thesis are given in this section. It should be noted that their selection is, to a certain extent, subjective. Some further mechanisms will be defined at the spots where they are needed.

For the sake of the present thesis, the context-free case is of main interest, that is, context-free grammars with controlled derivations, ((uniformly) limited) L systems rewriting symbols without any context dependencies (EOL systems and variants thereof) and grammar systems with context-free components. Nevertheless, we will present some definitions in a more general way so that they are suitable also for the next chapter about accepting grammars and systems.

In what follows, let  $V_N$  and  $V_T$  be two disjoint alphabets, the alphabet of nonterminals and the alphabet of terminal symbols, respectively. The union  $V_N \cup V_T$  is referred to as the

total alphabet and is denoted by  $V_G$ . Moreover, let  $S \in V_N$  be a designated nonterminal referred to as axiom.

**Definition 2.1 (Matrix grammars)** A *matrix grammar* ([1, 41]) is a quintuple  $G = (V_N, V_T, M, S, F)$ , where  $M$  is a finite set of matrices each of which is a finite sequence

$$m : (\alpha_1 \rightarrow \beta_1, \alpha_2 \rightarrow \beta_2, \dots, \alpha_n \rightarrow \beta_n),$$

$n \geq 1$ , of ‘usual’ rewriting rules over  $V_G$ , the core rules of  $G$ , and  $F$  is a finite set of occurrences of such rules in  $M$ . For  $x, y \in V_G^*$  and a matrix  $m : (\alpha_1 \rightarrow \beta_1, \alpha_2 \rightarrow \beta_2, \dots, \alpha_n \rightarrow \beta_n)$  in  $M$ , we write  $x \xrightarrow{m} y$  (or simply  $x \Rightarrow y$  if there is no danger of confusion) if and only if there are strings  $y_0, y_1, \dots, y_n$  such that  $y_0 = x$ ,  $y_n = y$ , and for  $1 \leq i \leq n$ , either

$$y_{i-1} = z_{i-1}\alpha_i z'_{i-1}, y_i = z_{i-1}\beta_i z'_{i-1} \text{ for some } z_{i-1}, z'_{i-1} \in V_G^*$$

or  $y_{i-1} = y_i$ , the rule  $\alpha_i \rightarrow \beta_i$  is not applicable to  $y_{i-1}$ , that is,  $\alpha_i \notin \text{Sub}(y_{i-1})$ , and the occurrence of  $\alpha_i \rightarrow \beta_i$  appears in  $F$ . One says that the rules whose occurrences appear in  $F$  are used in *appearance checking mode*, since, during a derivation according to a matrix in  $M$ , they can be passed over if not applicable. A matrix grammar is defined with (without) appearance checking if  $F \neq \emptyset$  ( $F = \emptyset$ , respectively). The language generated by  $G$  is defined as

$$L_{\text{gen}}(G) = \{ w \in V_T^* \mid S \xrightarrow{m_1} v_1 \xrightarrow{m_2} v_2 \xrightarrow{m_3} \dots \xrightarrow{m_k} v_k = w, k \geq 1, m_i \in M, 1 \leq i \leq k \}.$$

**Definition 2.2 (Programmed grammars)** A *programmed grammar* ([93, 41]) is a tuple  $G = (V_N, V_T, P, S)$ , where  $P$  is a finite set of productions of the form  $(r : \alpha \rightarrow \beta, \sigma(r), \varphi(r))$ , where  $r : \alpha \rightarrow \beta$  is a rewriting rule, called the core rule of the production, labeled by  $r$  (the labels uniquely determine productions, but there may be some productions with different labels having identical core rules  $\alpha \rightarrow \beta$ ), and  $\sigma(r)$  and  $\varphi(r)$  are two sets of labels of such core rules in  $P$ . By  $\text{Lab}(P)$ , we denote the set of all labels of the productions appearing in  $P$ .

A sequence of words  $y_0, y_1, \dots, y_n$  over  $V_G^*$  is referred to as a derivation in  $G$  if and only if, for  $1 \leq i \leq n$ , there are productions  $(r_i : \alpha_i \rightarrow \beta_i, \sigma(r_i), \varphi(r_i)) \in P$  such that either

$$y_{i-1} = z_{i-1}\alpha_i z'_{i-1}, y_i = z_{i-1}\beta_i z'_{i-1}, \text{ and, if } 1 \leq i < n, r_{i+1} \in \sigma(r_i)$$

or

$$\alpha_i \notin \text{Sub}(y_{i-1}), y_{i-1} = y_i, \text{ and, if } 1 \leq i < n, r_{i+1} \in \varphi(r_i).$$

In the latter case, the derivation step is done in *appearance checking mode*. The set  $\sigma(r_i)$  is called *success field* and the set  $\varphi(r_i)$  *failure field* of  $r_i$ . If  $\varphi(r) = \emptyset$  for all  $r \in \text{Lab}(P)$ , then  $G$  is said to be without appearance checking. We also write the derivation as

$$y_0 \xrightarrow{r_1} y_1 \xrightarrow{r_2} \dots \xrightarrow{r_n} y_n$$

or simply as  $y_0 \Rightarrow y_1 \Rightarrow \dots \Rightarrow y_n$ . Also the notations  $y_0 \xrightarrow{n} y_n$  and  $y_0 \xrightarrow{*} y_n$  are used. The language generated by  $G$  is defined as  $L_{\text{gen}}(G) = \{ w \in V_T^* \mid S \xrightarrow{*} w \}$ .

**Definition 2.3 (Random context grammars)** A *random context grammar* ([112, 41]) is a quadruple  $G = (V_N, V_T, P, S)$  where  $P$  is a finite set of random context rules, that is, triples of the form  $(\alpha \rightarrow \beta, Q, R)$  where  $\alpha \rightarrow \beta$  is a rewriting rule over  $V_G$  (the core rule of the



random context rule), and  $Q$  and  $R$  are subsets of  $V_N$ . For  $x, y \in V_G^*$ , we write  $x \Rightarrow y$  if and only if  $x = z_1\alpha z_2$ ,  $y = z_1\beta z_2$  for some  $z_1, z_2 \in V_G^*$ ,  $(\alpha \rightarrow \beta, Q, R)$  is a triple in  $P$ , all symbols of  $Q$  appear in  $z_1 z_2$ , and no symbol of  $R$  appears in  $z_1 z_2$ .  $Q$  is called the permitting context of  $\alpha \rightarrow \beta$  and  $R$  is the forbidding context of this rule; if  $R = \emptyset$  for all random context rules in  $P$ , then  $G$  is a random context grammar without appearance checking. The language generated by  $G$  is the set  $L_{\text{gen}}(G) = \{w \in V_T^* \mid S \xRightarrow{*} w\}$ .

**Definition 2.4 (Grammars with regular control)** Let  $G' = (V_N, V_T, P, S)$  be a type- $n$  grammar with labelled rules, that is,  $P = \{r_1 : \alpha_1 \rightarrow \beta_1, r_2 : \alpha_2 \rightarrow \beta_2, \dots, r_n : \alpha_n \rightarrow \beta_n\}$ , let  $\text{Lab}(P)$  be the set of all labels  $\{r_1, r_2, \dots, r_n\}$ ,  $F \subseteq \text{Lab}(P)$ , and let  $R$  be a regular language over the alphabet  $\text{Lab}(P)$ . Then  $G = (V_N, V_T, P, S, R, F)$  is referred to as a (type- $n$ ) *grammar with regular control* (cf. [58, 41, 99]). The language generated by  $G$  consists of all words  $w$  for which there is a word  $r_{i_1} r_{i_2} \dots r_{i_k}$  in  $R$  and there are strings  $x_0, x_1, \dots, x_k$  such that  $x_0 = S$  and  $x_k = w$  and, for  $1 \leq j \leq k$ , either

$$x_{j-1} = z_{j-1}\alpha_{i_j}z'_{j-1}, x_j = z_{j-1}\beta_{i_j}z'_{j-1} \text{ for some } z_{j-1}, z'_{j-1} \in V_G^*,$$

or  $x_{j-1} = x_j$ , the rule with label  $r_{i_j}$  is not applicable to  $x_{j-1}$ , and  $r_{i_j} \in F$ . The rules with a label in  $F$  are used in appearance checking mode. If  $F = \emptyset$ , then  $G$  is said to be without appearance checking.

**Definition 2.5 (Ordered grammars)** An *ordered grammar* ([56, 41]) is a quintuple  $G = (V_N, V_T, P, S, \prec)$ , where  $(V_N, V_T, P, S)$  is a phrase structure grammar and  $\prec$  is a partial order on  $P$ . A production  $\alpha \rightarrow \beta$  is applicable to a string  $x$  if  $x = z_1\alpha z_2$  for some  $z_1, z_2 \in V_G^*$  and  $x$  contains no subword  $\alpha'$  such that  $\alpha' \rightarrow \beta' \in P$  for some  $\beta'$  and  $\alpha \rightarrow \beta \prec \alpha' \rightarrow \beta'$ ; then  $x \Rightarrow y$  with  $y = z_1\beta z_2$ . As usual,  $L_{\text{gen}}(G) = \{w \in V_T^* \mid S \xRightarrow{*} w\}$ .

Thus, in ordered grammars only productions can be applied to a sentential form  $x$  which are maximal with respect to  $\prec$  among all productions with left-hand sides in  $\text{Sub}(x)$ .

**Definition 2.6 (Conditional grammars)** A *conditional grammar* ([85, 41]) is a pair  $(G, \rho)$ , where  $G = (V_N, V_T, P, S)$  is a phrase structure grammar and  $\rho$  is a mapping of  $P$  into the family of regular languages over  $V_G$ . For  $x, y \in V_G^*$ , we write  $x \Rightarrow y$  if and only if  $x = z_1\alpha z_2$ ,  $y = z_1\beta z_2$ ,  $z_1, z_2 \in V_G^*$ ,  $\alpha \rightarrow \beta \in P$ , and  $x \in \rho(\alpha \rightarrow \beta)$ . The language generated by a conditional grammar  $(G, \rho)$  is defined by  $L_{\text{gen}}(G, \rho) = \{w \in V_T^* \mid S \xRightarrow{*} w\}$ .

**Definition 2.7 (Lindenmayer systems)** An *extended tabled 0L system* (ETOL system) ([95, 41, 96]) is a quadruple  $G = (\Sigma, \Delta, \{P_1, P_2, \dots, P_r\}, \omega)$ , where  $\Delta$  is a non-empty subset of the alphabet  $\Sigma$ ,  $\omega \in \Sigma^+$  is the axiom, and each so-called table  $P_j$  is a finite subset of  $\Sigma \times \Sigma^*$  which satisfies the condition that, for each  $a \in \Sigma$ , there is a word  $w_a \in \Sigma^*$  such that  $(a, w_a) \in P_i$  (the elements of  $P_j$  are written as  $a \rightarrow w_a$  again), that is, each  $P_j$  defines a finite substitution  $\sigma_j : \Sigma^* \rightarrow 2^{\Sigma^*}$ ,  $\sigma_j(a) = \{w \mid a \rightarrow w \in P_j\}$ . We write  $x \Rightarrow y$  if and only if  $y \in \sigma_j(x)$  for some  $j$ , and  $L_{\text{gen}}(G) = \{v \in \Delta^* \mid \omega \xRightarrow{*} v\}$ , where  $\xRightarrow{*}$  is the reflexive and transitive closure of the relation  $\Rightarrow$ .

**Definition 2.8 (Exact uniformly-limited L systems)** An *exact  $k$ -uniformly-limited ETOL* (*kulETOL,ex*) *system* ([17]) is a tuple  $G = (\Sigma, \Delta, \{P_1, P_2, \dots, P_r\}, \{\omega_1, \omega_2, \dots, \omega_s\}, k)$ , where  $\Sigma$  is the total alphabet,  $\Delta \subseteq \Sigma$  is the terminal alphabet, each  $P_j$  is a table as in ETOL

systems,  $\{\omega_1, \omega_2, \dots, \omega_s\} \subset \Sigma^+$  is a finite set of axioms, and  $k$  is an integer with  $k \geq 1$ . The yield relation  $\Rightarrow$  is defined as follows:  $x \Rightarrow y$  if and only if there is a table  $P_j$  and, for  $1 \leq \kappa \leq k$ , there are productions  $a_\kappa \rightarrow w_\kappa \in P_j$  and words  $x_0, x_1, \dots, x_k$  such that  $x = x_0 a_1 x_1 \cdots a_k x_k$  and  $y = x_0 w_1 x_1 \cdots w_k x_k$ . The language generated by  $G$  is

$$L_{\text{gen}}(G) = \{w \in \Delta^* \mid \omega_\sigma \xRightarrow{*} w \text{ for some } \sigma \text{ with } 1 \leq \sigma \leq s\}.$$
<sup>2</sup>

Besides the definition of the yield relation given by an exact *kul*ETOL system, there is a definition by Wätjen [115], where each symbol in a word shorter than  $k$  is replaced (*kul*ETOL system). In view of accepting systems, which will be introduced in the next chapter, we prefer the following definition.

**Definition 2.9 (Uniformly limited L systems)** A *k-uniformly-limited ETOL system* (abbreviated as *kul*ETOL system) is a quintuple  $G = (\Sigma, \Delta, \{P_1, P_2, \dots, P_r\}, \{\omega_1, \omega_2, \dots, \omega_s\}, k)$ . The components are defined as for exact *kul*ETOL systems. For  $x \in \Sigma^*$ , let  $L_k(x)$  denote the set of strings derived directly from  $x$  via  $G$ , where  $G$  is interpreted as an exact *kul*ETOL system. Now define  $x \Rightarrow y$  if and only if  $y \in L_k(x)$  or there is a partition  $x = x_1 x_2 \cdots x_l$  with  $l \leq k$  and there is a table in  $G$  containing productions  $x_1 \rightarrow y_1, x_2 \rightarrow y_2, \dots, x_l \rightarrow y_l$  such that  $y = y_1 y_2 \cdots y_l$ .

Note that this definition coincides with Wätjen's definition.

As another concept of limited parallelism, *k-limited ETOL systems* have been defined in [55, 113], where  $k$  occurrences of any symbol of the alphabet are replaced simultaneously. Here, we present the formal definition only for the case  $k = 1$ . This will be sufficient for the objective of this thesis, particularly since any *k-limited ETOL system* can be simulated by a *1-limited ETOL system*,  $k \geq 1$  (regardless of whether or not erasing productions are allowed), see [49] and [113].

**Definition 2.10 (Limited L systems)** A *1-limited ETOL system* (abbreviated as *1l*ETOL system) is a quintuple  $G = (\Sigma, \Delta, \{P_1, P_2, \dots, P_r\}, \omega, 1)$  such that  $(\Sigma, \Delta, \{P_1, P_2, \dots, P_r\}, \omega)$  is an ETOL system. According to  $G$ ,  $x \Rightarrow y$  (for  $x, y \in \Sigma^*$ ) if and only if there is a table  $P_j$  and partitions  $x = x_0 a_1 x_1 \cdots a_n x_n$ ,  $y = x_0 w_1 x_1 \cdots w_n x_n$  such that  $a_\nu \rightarrow w_\nu \in P_j$  for each  $1 \leq \nu \leq n$ ,  $a_\nu \neq a_\mu$  for  $\nu \neq \mu$ , and each left-hand side  $z$  of a production in  $P_j$  is either equal to some  $a_\nu$  or not contained in  $\text{Sub}(x_0) \cup \text{Sub}(x_1) \cup \cdots \cup \text{Sub}(x_n)$ .

**Definition 2.11 (Scattered context grammars)** A *scattered context grammar* ([59, 41]) is a quadruple  $G = (\Sigma, \Delta, P, S)$ , where  $\Sigma$  and  $\Delta$  denote the total alphabet and the terminal alphabet, respectively, and  $S \in \Sigma \setminus \Delta$  is the axiom.  $P$  is a finite set of finite sequences of context-free productions,  $P = \{p_1, p_2, \dots, p_n\}$ , written as

$$p_i : (v_{i1}, v_{i2}, \dots, v_{ir_i}) \rightarrow (w_{i1}, w_{i2}, \dots, w_{ir_i}), \quad 1 \leq i \leq n.$$

The application of such a rule  $p_i$  to some  $x \in \Sigma^+$  yields  $y \in \Sigma^*$  (written as  $x \Rightarrow y$ ), if

$$x = x_0 v_{i1} x_1 v_{i2} \cdots x_{r_i} v_{ir_i} x_{r_i}, \quad \text{and} \quad y = x_0 w_{i1} x_1 w_{i2} \cdots x_{r_i} w_{ir_i} x_{r_i},$$

for some words  $x_0, x_1, \dots, x_{r_i}$ . Denoting the reflexive transitive closure of  $\Rightarrow$  by  $\xRightarrow{*}$ , we define  $L_{\text{gen}}(G) = \{w \in \Delta^* \mid S \xRightarrow{*} w\}$ .

---

<sup>2</sup>Obviously, apart from the concept of tables, this definition is equivalent to the definition of *k-context-free grammars* given by K. Salomaa in [100].

**Definition 2.12 (Unordered scattered context grammars)** An *unordered scattered context grammar* ([78, 41]) is defined as a scattered context grammar above, where the application of such a rule  $p_i$  to some  $x \in \Sigma^+$  yields  $y \in \Sigma^*$  (written as  $x \Rightarrow y$ ), if there is a permutation  $\pi : \{1, 2, \dots, r_i\} \rightarrow \{1, 2, \dots, r_i\}$  such that

$$x = x_1 v_{\pi(1)} x_2 v_{\pi(2)} \cdots x_{r_i} v_{\pi(r_i)} x_{r_i+1}, \quad \text{and} \quad y = x_1 w_{\pi(1)} x_2 w_{\pi(2)} \cdots x_{r_i} w_{\pi(r_i)} x_{r_i+1},$$

for some words  $x_0, x_1, \dots, x_{r_i}$ . Again,  $L_{\text{gen}}(G) = \{w \in \Delta^* \mid S \xRightarrow{*} w\}$ , where  $\xRightarrow{*}$  denotes the reflexive, transitive closure of the yield relation  $\Rightarrow$ .

Next, we turn to the definition of grammar systems, where we restrict ourselves to systems with context-free productions.

**Definition 2.13 (Cooperating distributed grammar systems)** A *cooperating distributed grammar system* (CD grammar system) ([31, 32, 43]) with  $n$  context-free components is a tuple  $\Gamma = (V_N, V_T, S, P_1, P_2, \dots, P_n)$  such that  $G_i = (V_N, V_T, P_i, S)$  is a context-free grammar, for  $1 \leq i \leq n$ . The total alphabet  $V_N \cup V_T$  of  $\Gamma$  is denoted by  $V_\Gamma$ . The set  $\text{dom}(P_i) = \{A \in V_N \mid A \rightarrow z \in P_i\}$  is called the *domain* of  $P_i$ . Furthermore, we set  $T_i = V_\Gamma \setminus \text{dom}(P_i)$ ,  $1 \leq i \leq n$ . Let  $x, y \in V_\Gamma^*$  and  $k$  be a positive integer. For  $1 \leq i \leq n$ , we write  $x \Rightarrow_i y$  if  $x \Rightarrow y$  according to the grammar  $G_i$ . Hence, subscript  $i$  refers to the context-free grammar (the component) to be used.

- A *\*-derivation* by the  $i$ th component is defined by  $x \xRightarrow{*}_i y$ , where  $\xRightarrow{*}_i$  denotes the reflexive and transitive closure of  $\Rightarrow_i$ .
- A *k-step derivation* by the  $i$ th component is defined by

$$x \xRightarrow{k}_i y \text{ iff } x = x_0 \Rightarrow_i x_1 \Rightarrow_i x_2 \Rightarrow_i \dots \Rightarrow_i x_k = y,$$

for some strings  $x_1, x_2, \dots, x_{k-1}$  in  $V_\Gamma^*$ .

- An *at most k-step derivation* by the  $i$ th component is defined by

$$x \xRightarrow{\leq k}_i y \text{ iff } x \xRightarrow{k'}_i y \text{ for some } k' \leq k.$$

- An *at least k-step derivation* by the  $i$ th component is defined by

$$x \xRightarrow{\geq k}_i y \text{ iff } x \xRightarrow{k'}_i y \text{ for some } k' \geq k.$$

- A *t-derivation* by the  $i$ th component is defined by

$$x \xRightarrow{t}_i y, \text{ iff } x \xRightarrow{*}_i y, \text{ and there is no } z \text{ with } y \Rightarrow_i z.$$

- A *derivation in the full-competence mode* by the  $i$ th component is defined by

$$x \xRightarrow{\text{full}}_i y, \text{ iff } x = x_0 \Rightarrow_i x_1 \Rightarrow_i x_2 \Rightarrow_i \dots \Rightarrow_i x_m = y,$$

where for all  $j$ ,  $0 \leq j \leq m - 1$ , we have

$$x_j \in (\text{dom}(P_i) \cup V_T)^*, \text{ and } \text{symb}(y) \cap (V_N \setminus \text{dom}(P_i)) \neq \emptyset \text{ or } y \in V_T^*.$$

The language generated by the CD grammar system  $\Gamma$  in the mode  $\gamma$  of derivation,  $\gamma \in \{*, t, \text{full}\} \cup \{=k, \leq k, \geq k \mid k \geq 1\}$ , is

$$L(\Gamma, \gamma) = \{w \in V_{\Gamma}^* \mid S \xrightarrow{\gamma}_{i_1} w_1 \xrightarrow{\gamma}_{i_2} w_2 \dots \xrightarrow{\gamma}_{i_m} w_m = w, \\ \text{where } m \geq 1, 1 \leq i_j \leq n, w_j \in V_{\Gamma}^*, 1 \leq j \leq m\}.$$

Intuitively, in each derivation mode, the component which starts rewriting the sentential form is selected in a nondeterministic manner. The derivation mode determines the condition when a component, once started, stops rewriting and “hands over” the sentential form to the next (not necessarily different) component. In particular, in the  $t$ -mode of derivation, a component stops rewriting if and only if it has no production left which is applicable. In the full competence mode, a component stops if and only if there is one nonterminal symbol present in the sentential form which cannot be rewritten. The full competence mode has been defined in [75] and has been called *sf-mode* in [11] and some succeeding papers; the  $\geq k$ -mode is from [42], and all the other modes have been introduced in [31].

**Definition 2.14 (Parallel communicating grammar systems)** A *parallel communicating grammar system* (PC grammar system) ([91, 32, 43]) with  $n$  context-free components,  $n \geq 1$ , is an  $(n + 3)$ -tuple  $\Gamma = (N, K, T, G_1, G_2, \dots, G_n)$ , where  $N$ ,  $K$ , and  $T$  are pairwise disjoint alphabets of nonterminal symbols, query symbols, and terminal symbols, respectively. For  $1 \leq i \leq n$ ,  $G_i = (N \cup K, T, P_i, S_i)$  is a context-free grammar with nonterminal alphabet  $N \cup K$ , terminal alphabet  $T$ , a set of rewriting rules  $P_i \subseteq (N \cup K) \times (N \cup K \cup T)^*$  and an axiom  $S_i$ .

The grammars  $G_1, G_2, \dots, G_n$  are called components of  $\Gamma$ ,  $G_1$  is said to be the master component (or master grammar) of  $\Gamma$ . The total alphabet  $N \cup K \cup T$  of  $\Gamma$  is denoted by  $V_{\Gamma}$ .

Let  $\Gamma = (N, K, T, G_1, G_2, \dots, G_n)$ ,  $n \geq 1$ , be a PC grammar system as above. An  $n$ -tuple  $(x_1, x_2, \dots, x_n)$ , where  $x_i \in V_{\Gamma}^*$ ,  $1 \leq i \leq n$ , is called a configuration of  $\Gamma$ . The configuration  $(S_1, S_2, \dots, S_n)$  is said to be the initial configuration.

PC grammar systems change their configurations by performing direct derivation steps. Let  $\Gamma = (N, K, T, G_1, G_2, \dots, G_n)$ ,  $n \geq 1$ , be a PC grammar system, and let  $(x_1, x_2, \dots, x_n)$  and  $(y_1, y_2, \dots, y_n)$  be two configurations of  $\Gamma$ . We say that  $(x_1, x_2, \dots, x_n)$  directly derives  $(y_1, y_2, \dots, y_n)$ , denoted by  $(x_1, x_2, \dots, x_n) \Longrightarrow (y_1, y_2, \dots, y_n)$ , if one of the next two cases holds:

1. If there is no  $x_i$  which contains any query symbol, then, for  $1 \leq i \leq n$ , either  $x_i \in (N \cup T)^* \setminus T^*$  and  $x_i \xrightarrow{G_i} y_i$ , or  $x_i \in T^*$  and  $y_i = x_i$ .

2. There is some  $x_i$ ,  $1 \leq i \leq n$ , which contains at least one occurrence of query symbols. Let  $x_i$  be of the form  $x_i = z_1 Q_{i_1} z_2 Q_{i_2} \dots z_t Q_{i_t} z_{t+1}$ , where  $z_j \in (N \cup T)^*$ ,  $1 \leq j \leq t + 1$ , and  $Q_{i_l} \in K$ ,  $1 \leq l \leq t$ . In this case  $y_i = z_1 x_{i_1} z_2 x_{i_2} \dots z_t x_{i_t} z_{t+1}$ , if  $x_{i_l}$ ,  $1 \leq l \leq t$ , does not contain any query symbol. In so-called returning systems,  $y_{i_l} = S_{i_l}$ , for  $1 \leq l \leq t$ . In non-returning systems  $y_{i_l} = x_{i_l}$ ,  $1 \leq l \leq t$ . If some  $x_{i_l}$  contains at least one occurrence of query symbols, then  $y_i = x_i$ . For all  $i$ ,  $1 \leq i \leq n$ , for which  $y_i$  is not specified above,  $y_i = x_i$  holds.

The first case is the description of a rewriting step: if no query symbols are present in any of the sentential forms, then each component grammar uses one of its rewriting rules except those which have already produced a terminal string. The derivation is blocked if a sentential form is not a terminal string, but no rule can be applied to it.

The second case describes a communication step which has priority over effective rewriting: if some query symbol, say  $Q_j$ , appears in a sentential form, then the rewriting process stops

and a communication step must be performed. The symbol  $Q_j$  has to be replaced by the current sentential form of component  $G_j$ , say  $x_j$ , supposing that  $x_j$  does not contain any query symbol. If this sentential form also contains query symbols, then first these symbols must be replaced with the requested sentential forms and so on. If this condition cannot be fulfilled (a circular query appeared), then the derivation is blocked.

If the sentential form of a component was communicated to another one, this component can continue its own work in two ways: in so-called *returning* systems, the component must return to its sentential form of the initial configuration (the axiom) and begin to derive a new string. In *non-returning* systems the components do not return to their initial sentential forms but continue to process their current string.

In the following, by  $\xRightarrow{*}$  the reflexive and transitive closure of the yield relation  $\Rightarrow$  is denoted.

Let  $\Gamma = (N, K, T, G_1, G_2, \dots, G_n)$  be a PC grammar system as above. The language of  $\Gamma$  is defined by

$$L_{\text{gen}}(\Gamma) = \{x_1 \in T^* \mid (S_1, S_2, \dots, S_n) \xRightarrow{*} (x_1, x_2, \dots, x_n), x_i \in V_{\Gamma}^*, 2 \leq i \leq n\}.$$

Thus, the generated language consists of the terminal strings appearing as sentential forms of the master grammar  $G_1$  in a derivation which started off with the initial configuration  $(S_1, S_2, \dots, S_n)$ .

Finally, we introduce the following notation in order to refer to the classes of grammars which have been listed above. Let  $X \in \{\text{REG}, \text{LIN}, \text{CF}-\lambda, \text{CF}, \text{CS}, \text{RE}\}$ . By  $(M, X, \text{ac})$ ,  $(P, X, \text{ac})$ ,  $(\text{RC}, X, \text{ac})$  and  $(\text{rC}, X, \text{ac})$  we denote the classes of all matrix grammars, programmed grammars, random context grammars and grammars with regular control, respectively, with core rules of type  $X$ . If the grammars are restricted to be without appearance checking, then ‘ac’ is omitted in that notation. Correspondingly,  $(O, X)$ ,  $(C, X)$ ,  $(\text{SC}, X)$  and  $(\text{uSC}, X)$  are the classes of all ordered, conditional, scattered context and unordered scattered context grammars, respectively, with rewriting rules of type  $X$ .

Furthermore,  $\text{ETOL}$ ,  $(k\text{uETOL}, \text{ex})$ ,  $k\text{uETOL}$  and  $k\text{ETOL}$  denote the classes of the corresponding Lindenmayer systems,  $k \geq 1$ . Furthermore, we define

$$([\text{u}]k\text{ETOL}[\text{ex}]) = \bigcup_{k \geq 1} (k[\text{u}]k\text{ETOL}[\text{ex}]).^3$$

Moreover, if in the systems  $G = (\Sigma, \Delta, \{P_1, P_2, \dots, P_r\}, \omega)$  no table  $P_j$  contains a  $\lambda$ -rule, that is,  $P_j \subseteq \Sigma \times \Sigma^+$  for each  $j$ , we call the systems propagating; in this case, we add the letter P to the notation, leading to the class  $\text{EPTOL}$  and its limited variants. Finally, in case of  $r = 1$ , the letter T is omitted in the notation of the classes. Clearly, these restrictions may be combined.

Let  $X$  be a class of devices as above. Then  $\mathcal{L}(X)$  is written for the family of languages which can be generated by some device from  $X$ .

As to grammar systems, let  $\mathcal{L}(\text{CD}_n, \text{CF}, \gamma)$  denote the family of languages generated by CD grammar systems with  $n$  context-free components in the mode  $\gamma$  of derivation,  $\gamma \in \{*, \text{t}, \text{full}\} \cup \{=k, \leq k, \geq k \mid k \geq 1\}$ , and let  $\mathcal{L}(\text{CD}_{\infty}, \text{CF}, \gamma) = \bigcup_{n \geq 1} \mathcal{L}(\text{CD}_n, \text{CF}, \gamma)$ . In this notation, CF is replaced with  $\text{CF}-\lambda$  if systems having  $\lambda$ -rules in at least one of the components are excluded.

<sup>3</sup>Whenever we use bracket notations like this, we mean that the statement is true both in case of neglecting the bracket contents and in case of ignoring the brackets themselves.

Finally, we shall denote the families of languages of returning and non-returning PC grammar systems with context-free components by  $\mathcal{L}(\text{PC}_\infty\text{CF})$  and  $\mathcal{L}(\text{NPC}_\infty\text{CF})$ , respectively. When only centralized PC grammar systems are used, we add the letter C coming to the families  $\mathcal{L}(\text{CPC}_\infty\text{CF})$  and  $\mathcal{L}(\text{NCPC}_\infty\text{CF})$ . We replace CF by CF- $\lambda$  in that notation if  $\lambda$ -rules are forbidden in any component of the systems.

## 2.2 Generative Capacities

In this section the hierarchical relationships between the families of languages generated by the devices *with context-free core rules* presented in the preceding section are given. Moreover, we locate these families in the Chomsky hierarchy. Further properties such as closure or decidability properties can be found in the standard references [32, 41, 43, 44, 96].

In some sense, the following diagrams supplement the diagram on page 146 in [41] and Theorem 2.15 including the figure on page 137 in [44]; see also [17]. In the diagrams given below, solid lines indicate strict inclusion, where the larger language class is near the arrow-tip, dashed lines indicate an inclusion relation where the strictness is unknown, and dotted lines mean that the inclusion relation which is indicated by the arrow tip cannot hold.

Below, we only list proofs of results not contained in the standard references [41, 96, 97].

### 2.2.1 Single Devices: The Non-Erasing Case

The following equalities are valid (see [44]):

1.  $\mathcal{L}(\text{P}, \text{CF}-\lambda) = \mathcal{L}(\text{M}, \text{CF}-\lambda) = \mathcal{L}(\text{rC}, \text{CF}-\lambda) = \mathcal{L}(\text{uSC}, \text{CF}-\lambda)$
2.  $\mathcal{L}(\text{RC}, \text{CF}-\lambda) \subseteq \mathcal{L}(\text{P}, \text{CF}-\lambda)$
3.  $\mathcal{L}(\text{P}, \text{CF}-\lambda, \text{ac}) = \mathcal{L}(\text{M}, \text{CF}-\lambda, \text{ac}) = \mathcal{L}(\text{RC}, \text{CF}-\lambda, \text{ac}) = \mathcal{L}(\text{rC}, \text{CF}-\lambda, \text{ac})$
4.  $\mathcal{L}(\text{C}, \text{CF}-\lambda) = \mathcal{L}(\text{CS})$

Moreover, the relations shown in Figure 2.1 hold.

Reference of claims:

- In [60], it is shown that  $\{a^{2^n} \mid n \in \mathbb{N}\} \notin \mathcal{L}(\text{P}, \text{CF})$ . Hence, there are EP0L languages which are not in  $\mathcal{L}(\text{P}, \text{CF})$ . This naturally applies also to superfamilies of  $\mathcal{L}(\text{EPT0L})$  and to subfamilies of  $\mathcal{L}(\text{P}, \text{CF})$ . Especially, this partially answers a question raised in [115] concerning the relationship of uniformly limited L systems and L systems.
- The same example shows the strictness of the inclusion between  $\mathcal{L}(\text{P}, \text{CF}-\lambda)$  and  $\mathcal{L}(\text{P}, \text{CF}-\lambda, \text{ac})$ .
- In [47, Theorem 5.2], it is shown that  $\mathcal{L}(\text{O}, \text{CF}-\lambda) \subset \mathcal{L}(\text{1IEPT0L})$  which improves [48, Lemma 3.7].
- The strictness of the inclusion  $\mathcal{L}(\text{CF}-\lambda) \subset \mathcal{L}(\text{uIEP0L})$  can be seen by Example 2.1 in [115], where also the incomparability results concerning EP0L are proved.
- In [113, Theorem 4.4] together with [114], it is shown that  $\mathcal{L}(\text{IEP0L})$  does not contain  $\mathcal{L}(\text{EP0L})$ . Similarly, [114] shows the strictness of the inclusion  $\mathcal{L}(\text{IEP0L}) \subset \mathcal{L}(\text{IEPT0L})$ .

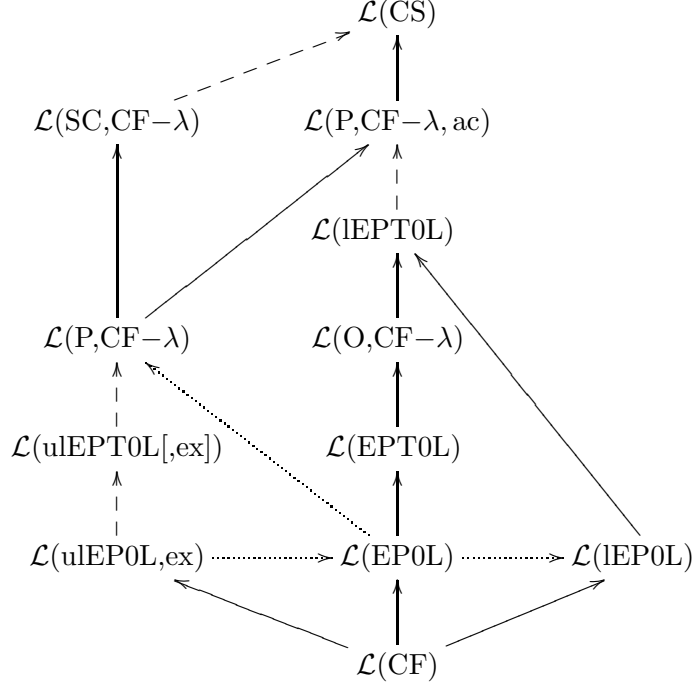


Figure 2.1: Hierarchy of families of languages generated by single devices without erasing rules.

- The inclusion  $\mathcal{L}(\text{IEPT0L}) \subseteq \mathcal{L}(\text{P}, \text{CF}-\lambda, \text{ut})$  is proved in [37, 46]. The other inclusion is seen via a modification of the technique given in [37] in the presence of erasing productions.

Finally, the problem remains where to place the chain

$$\mathcal{L}(\text{CF}) \longrightarrow \mathcal{L}(\text{RC}, \text{CF}-\lambda) \dashrightarrow \mathcal{L}(\text{P}, \text{CF}-\lambda)$$

within the sketched diagram.

### 2.2.2 Single Devices: The Erasing Case

The following equalities are valid (see [44]):

1.  $\mathcal{L}(\text{P}, \text{CF}) = \mathcal{L}(\text{M}, \text{CF}) = \mathcal{L}(\text{rC}, \text{CF}) = \mathcal{L}(\text{uSC}, \text{CF}) = \mathcal{L}(\text{C}, \text{CF})$
2.  $\mathcal{L}(\text{RC}, \text{CF}) \subseteq \mathcal{L}(\text{P}, \text{CF})$
3.  $\mathcal{L}(\text{P}, \text{CF}, \text{ac}) = \mathcal{L}(\text{M}, \text{CF}, \text{ac}) = \mathcal{L}(\text{RC}, \text{CF}, \text{ac}) = \mathcal{L}(\text{rC}, \text{CF}, \text{ac})$   
 $\quad = \mathcal{L}(\text{SC}, \text{CF}) = \mathcal{L}(\text{C}, \text{CF}) = \mathcal{L}(\text{RE})$

Moreover, the relations shown in Figure 2.2 hold.

With the references given in the preceding subsection, most of the connections given in the following diagram are be clear. We add only the following few references:

- The equality  $\mathcal{L}(\text{SC}, \text{CF}) = \mathcal{L}(\text{RE})$  is shown in [74].

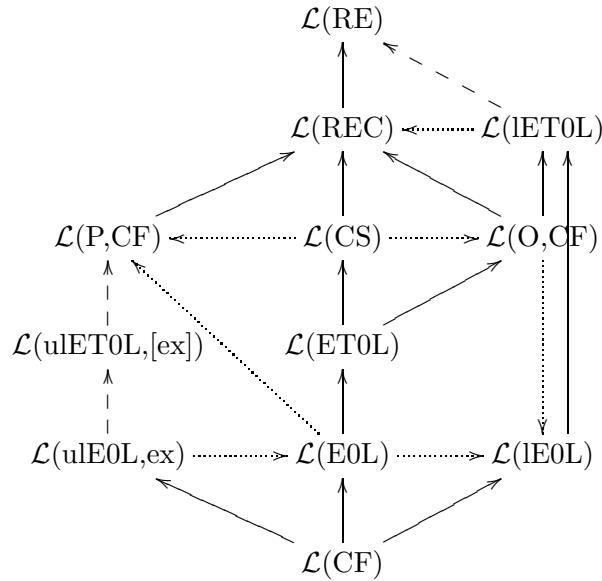


Figure 2.2: Hierarchy of families of languages generated by single devices when erasing rules are allowed.

- The strictness of the inclusion  $\mathcal{L}(1EOL) \subset \mathcal{L}(1ETOL)$  was shown in [114].
- The arguments for the equivalence between exact and non-exact ulETOL systems have been given in [52].
- We showed in [16] that ordered grammars have a solvable membership problem. Furthermore,  $\mathcal{L}(O,CF)$  is closed under homomorphism. If each context-sensitive language were generable by an ordered grammar, then also any homomorphic image of a context-sensitive language were generable by an ordered grammar, which would finally imply the recursiveness of any enumerable language.
- A similar argument is valid for  $\mathcal{L}(P,CF)$ .

Finally, the problem remains where to place the chain

$$\mathcal{L}(CF) \longrightarrow \mathcal{L}(RC,CF) \dashrightarrow \mathcal{L}(P,CF)$$

within the sketched diagram.

By definition, the erasing variants of all the grammar formalisms are at least as powerful as the non-erasing variants. The results listed above induce that the inclusions of the corresponding language families are strict for matrix, programmed and random context grammars as well as grammars with regular control (all with appearance checking), furthermore for conditional and scattered context grammars. The problem remains unsolved for matrix and programmed grammars without appearance checking, for grammars with regular control without appearance checking as well as for ordered grammars and unordered scattered context grammars, though recently some contributions towards a solution of these open questions have been made by showing that erasing productions can be removed in random context



grammars without appearance checking [118] and proving a sufficient condition for erasing productions to be avoidable in grammars with controlled derivations [119]. Moreover, in [120] a reformulation of this question for matrix grammars has been given in terms of Petri net controlled grammars. In the case of ET0L systems, erasing productions can be eliminated by a non-straightforward modification of the standard construction for context-free grammars, see [96]. For the limited cases it has been shown in [47] that  $\mathcal{L}(11EPT0L) \subset \mathcal{L}(11ET0L)$ .

### 2.2.3 Grammar Systems

The power of CD grammar systems depends on the derivation mode. The following is known [43].

- (i)  $\mathcal{L}(CD_\infty, CF[-\lambda], \gamma) = \mathcal{L}(CF)$ , for  $\gamma \in \{=1, \geq 1, *\} \cup \{\leq k \mid k \geq 1\}$ .
- (ii)  $\mathcal{L}(CF) = \mathcal{L}(CD_1, CF[-\lambda], \gamma) \subset \mathcal{L}(CD_2, CF[-\lambda], \gamma) \subseteq \mathcal{L}(CD_r, CF[-\lambda], \gamma)$   
 $\subseteq \mathcal{L}(CD_\infty, CF[-\lambda], \gamma) \subseteq \mathcal{L}(M, CF[-\lambda])$ , for  $\gamma \in \{=k, \geq k \mid k \geq 2\}$ ,  $r \geq 3$ .
- (iii)  $\mathcal{L}(CD_r, CF[-\lambda], =k) \subseteq \mathcal{L}(CD_r, CF[-\lambda], =sk)$ , for  $k, r, s \in \mathbb{N}$ .
- (iv)  $\mathcal{L}(CD_r, CF[-\lambda], \geq k) \subseteq \mathcal{L}(CD_r, CF[-\lambda], \geq(k+1))$ , for  $k, r \in \mathbb{N}$ .
- (v)  $\mathcal{L}(CF) = \mathcal{L}(CD_1, CF[-\lambda], t) = \mathcal{L}(CD_2, CF[-\lambda], t)$   
 $\subset \mathcal{L}(CD_3, CF[-\lambda], t) = \mathcal{L}(CD_\infty, CF[-\lambda], t) = \mathcal{L}(ET0L)$ .

Concerning the full-competence mode of derivation, the following inclusion chain is valid according to [75, 10].

- (vi)  $\mathcal{L}(CF) = \mathcal{L}(CD_1, CF[-\lambda], \text{full}) \subset \mathcal{L}(CD_2, CF[-\lambda], \text{full})$   
 $= \mathcal{L}(CD_3, CF[-\lambda], \text{full}) = \mathcal{L}(CD_\infty, CF[-\lambda], \text{full}) = \mathcal{L}(M, CF[-\lambda], \text{ac})$ .<sup>4</sup>

About PC grammar systems, one can find the following results in [32, 43, 109].

- (i)  $\mathcal{L}(PC_\infty CF-\lambda) = \mathcal{L}(NPC_\infty CF-\lambda) = \mathcal{L}(PC_\infty CF) = \mathcal{L}(NPC_\infty CF) = \mathcal{L}(RE)$
- (ii)  $\mathcal{L}(CF) \subset \mathcal{L}([N]CPC_\infty CF-\lambda) \subseteq \mathcal{L}(CS)$
- (iii)  $\mathcal{L}(CF) \subset \mathcal{L}([N]CPC_\infty CF) \subseteq \mathcal{L}(RE)$

Detailed proofs of results not contained in [32] can be found in [108] (which is an improvement of [45]), [33], and [70].

---

<sup>4</sup>In [10], the  $\lambda$ -free case has not been considered explicitly. However, in the construction of a CD grammar system with three components from an arbitrary one, no new erasing productions are introduced. Hence, the constructed grammar system is  $\lambda$ -free if the given one is so.

## Chapter 3

# Accepting Grammars and Systems

While programming (and many other) languages are usually defined via grammars, that is language *generating* devices, parsers are essentially language acceptors. As already Salomaa pointed out in [99], any device that generates words (hence describing a formal language) can be interpreted as a system that accepts words. A grammar, interpreted as language acceptor, starts its derivation with a terminal string (the sentence to be analyzed) and strives to derive the axiom, which serves as the “goal symbol” (instead of the “start symbol”), by applying its rewriting rules backwards, that is, when the left-hand sides and the right-hand sides are interchanged. Those *accepting grammars* still keep the feature of natural language descriptors but allow to be viewed as nondeterministic “semi-algorithmic parsers” halting on those input words which belong to the accepted language, yielding a parse if the syntactic correctness of the input is given. If the grammar formalism guarantees that the length of the words cannot increase during the derivations, then these nondeterministic acceptors can be realized as backtrack parsing algorithms as described in [3].

For the usually considered type- $n$  grammars of the Chomsky hierarchy, it does not matter whether these devices are considered in generating or accepting mode, in the sense that the family of languages generable by type- $n$  grammars equals the family of languages acceptable by type- $n$  grammars, see [99, Theorem I.2.1]. However, the trivial proof of this observation does not transfer to several cases of more involved language defining mechanisms.

In this chapter, we investigate grammar formalisms suitable for the description of non-context-free languages as those which are listed in Chapter 2. Indeed, we find representatives for each of the following cases:

- There is a trivial equivalence between accepting and generating mode, that is, we observe  $x \Rightarrow y$  in generating mode of a grammar  $G$  if and only if  $y \Rightarrow x$  in accepting mode of some (dual) grammar  $G'$ .
- We get equivalence between accepting and generating mode via a more complicated construction.
- Accepting and generating mode yield different language classes.

For many devices  $X$  considered in this thesis, the definitions of the generating devices given in the preceding chapters equally apply to the accepting case. We have covered both the generating and the accepting case, whenever these notions are inherited via the underlying Chomsky grammar. For the definition of the accepting mode of the devices, the general idea is the next:

- Instead of start words, we have goal words. We use the notion “axioms” both in generating and in accepting case.
- Generally, we allow only productions of a special form in generative devices. Since they turn out to be the most interesting case, mainly context-free productions of the form  $a \rightarrow w$ , where  $a$  is some symbol and  $w$  is some (possibly empty) word, are considered. In accepting mode, we turn these restrictions “around”, coming to productions of the form  $w \rightarrow a$  in the context-free case. Especially, accepting  $\lambda$ -productions are of the form  $\lambda \rightarrow a$ .

We call an accepting grammar  $G^d$  derived from a generating grammar  $G$  *dual* to  $G$  if  $G^d$  is obtained from  $G$  by interpreting start words as goal words and productions of the form  $v \rightarrow w$  as productions  $w \rightarrow v$ . Similarly, one can consider the dual  $H^d$  of an accepting grammar  $H$ . Obviously,  $(G^d)^d = G$  and  $(H^d)^d = H$ .

- An essential thing about grammars is their dynamic interpretation via the yield relation  $\Rightarrow$  (and its reflexive transitive closure  $\Rightarrow^*$ ). Following [15, 16, 52], we introduce the corresponding yield relation (also denoted by  $\Rightarrow$ ) of the accepting mode with textually the same words as in the generating case.

Observe that, when defining accepting counterparts of existing generating devices, we do not want to define accepting grammars just in order to mimic the generating steps the other way round, but we want to carry over the original idea and motivation of the generating mechanism in order to define the corresponding accepting mechanism. Formally, such accepting grammars look like their generating counterparts, just turning the core productions “around” and keeping the control mechanism ruling the application of these productions textually the same.

If needed, we will give more details about the accepting devices further below.

### 3.1 Accepting versus Generating Mode

In this section, we compare, for the various grammar formalisms, their power as generating and as accepting devices. If  $X$  is some device,  $L_{\text{gen}}(X)$  ( $L_{\text{acc}}(X)$ ) denotes the language generated (accepted) by the device  $X$ . If  $X$  is some class of devices (such as, for example,  $(M, CF)$ ),  $\mathcal{L}^{\text{gen}}(X)$  ( $\mathcal{L}^{\text{acc}}(X)$ ) denotes the family of languages each of which can be generated (accepted) by some device in  $X$ .<sup>1</sup> As in the preceding chapters, let  $V_N$  and  $V_T$  be two disjoint alphabets, the alphabet of nonterminals and the alphabet of terminal symbols, respectively. Moreover, we denote the total alphabet  $V_N \cup V_T$  of  $G$  by  $V_G$ .

#### 3.1.1 Context Condition Grammars

We already mentioned that, for type- $n$  grammars, the descriptive power of the generating and the accepting mode coincides. More precisely, if  $G$  is a generating type- $n$  grammar, then its dual  $G^d$  is an accepting type- $n$  grammar such that  $L_{\text{gen}}(G) = L_{\text{acc}}(G^d)$ , and vice versa. In this case, we find  $x \Rightarrow y$  in  $G$  if and only if  $y \Rightarrow x$  in  $G^d$ .

---

<sup>1</sup>For a device  $X$ ,  $L_{\text{gen}}(X)$  has been written as  $L(X)$  in the previous chapter, and for a class  $X$  of devices,  $\mathcal{L}^{\text{gen}}(X)$  is identical with  $\mathcal{L}(X)$ . The subscript ‘gen’ is only added to explicitly point to fact that the devices are treated in generating mode.

In this section, we introduce a type of grammar referred to as *context condition grammar* (cc grammar, for short) for which the trivial relation between say generating grammars and their dual accepting counterparts is true, as well. Since cc grammars generalize phrase structure grammars, we show the well-known result once more.

**Definition 3.1 (context condition grammars [17])** A *context condition (cc) grammar* is given by a construct  $G = (V_N, V_T, P, \Omega)$ , where  $P$  is a finite set of production tables  $P = \{P_1, P_2, \dots, P_t\}$  and  $\Omega \subset V_G^+$  is the finite set of axioms. Each table contains a finite number of productions  $p_{ij}$  of the form  $(v_{ij} \rightarrow w_{ij}, g_{ij})$  (with  $1 \leq i \leq t$  and  $1 \leq j \leq |P_i|$ ), where  $v_{ij}, w_{ij} \in V_G^+$  and  $g_{ij} \subseteq (V_G \cup \{\#\})^* \times \mathbb{N}$  (where  $\# \notin V_G$  is a new limiting symbol). As an exception, we admit  $\lambda$ -productions, more precisely, productions of the form  $(v_{ij} \rightarrow \lambda, g_{ij})$  in generating mode or productions of the form  $(\lambda \rightarrow w_{ij}, g_{ij})$  in accepting mode, respectively. We call  $v_{ij} \rightarrow w_{ij}$  the core production of the production  $p_{ij}$ . A table  $P_i$  is applied to a string  $x$  using the following steps:

1. Consider the string  $x$  to be partitioned into  $x = x_1v_1x_2v_2 \cdots x_nv_nx_{n+1}$ . Define  $x^\# = x_1\#x_2\# \cdots x_n\#x_{n+1}$ .
2. Select  $n$  productions  $p_{ij_1}, p_{ij_2}, \dots, p_{ij_n}$  from  $P_i$  (possibly with  $p_{ij_r} = p_{ij_s}$ ) such that for all  $1 \leq k \leq n$ :  $v_k = v_{ij_k}$  and  $(x^\#, k) \in g_{ij_k}$ .
3. Replace  $x$  using the selected productions such that we get the sentential form  $y = x_1w_{ij_1}x_2w_{ij_2} \cdots x_nw_{ij_n}x_{n+1}$ .

If we succeed going through these three steps, then we write  $x \Rightarrow y$  in this case. By  $\overset{*}{\Rightarrow}$ , we denote the reflexive and transitive closure of the relation  $\Rightarrow$ . The language generated by the cc grammar  $G$  is  $L_{\text{gen}}(G) = \{z \in V_T^* \mid (\exists \omega \in \Omega) \omega \overset{*}{\Rightarrow} z\}$ . The language accepted by the cc grammar  $G$  is  $L_{\text{acc}}(G) = \{z \in V_T^* \mid (\exists \omega \in \Omega) z \overset{*}{\Rightarrow} \omega\}$ .

Observe that cc grammars are a very broad framework, maybe comparable to selective substitution grammars [41]. Especially, note that we do not exclude the further derivation of terminal symbols. Hence, we incorporate pure rewriting<sup>2</sup>, too, and also the usual conventions in parallel rewriting.

**Example 3.1 ([17])** Every phrase structure grammar corresponds to an equivalent cc grammar  $G$  with one table and one one-letter-axiom letting  $g = V_G^*\{\#\}V_G^* \times \{1\}$  for any production. Conversely, to every cc grammar  $G = (V_N, V_T, \{P\}, \{\omega\})$  with one table with productions of the form  $(v \rightarrow w, V_G^*\{\#\}V_G^* \times \{1\})$  with  $|v|_{V_N} > 0$  in the generating or  $|w|_{V_N} > 0$  in the accepting case, respectively, and one one-letter axiom  $\omega \in V_N$ , there corresponds an equivalent type-0 grammar  $(V_N, V_T, P', \omega)$ . This is also true if we restrict ourselves to so-called left derivations, where we let  $g = V_T^*\{\#\}V_G^* \times \{1\}$  for any production. For context-sensitive grammars, it is possible to give another different characterization. Let  $H = (V_N, V_T, P, S)$  be a generating context-sensitive grammar. For each production  $y_1Ay_2 \rightarrow y_1wy_2$ , we put a production  $(A \rightarrow w, g)$  into the table  $P'$  of the equivalent cc grammar  $(V_N, V_T, \{P'\}, \{S\})$ , where  $(x_1\#x_2, n) \in g$  if and only if  $y_1 \in \text{Suf}(x_1)$  and  $y_2 \in \text{Pref}(x_2)$  and  $n = 1$ . Dually, we may treat the case of accepting context-sensitive grammars.

<sup>2</sup>In pure grammars and systems, there is no distinction between terminal and nonterminal symbols. As in the case of Lindenmayer systems, there is a total alphabet about which the rewriting rules are defined.

In the following, we formulate the concept of dual grammar formally for cc grammars. If  $G = (V_N, V_T, \{P_1, P_2, \dots, P_t\}, \Omega)$  is a cc grammar, then  $G^d = (V_N, V_T, \{P_1^d, P_2^d, \dots, P_t^d\}, \Omega)$  is called *dual* to  $G$  if  $P_i^d = \{(w \rightarrow v, g) \mid (v \rightarrow w, g) \in P_i\}$ . Obviously,  $(G^d)^d = G$ .

Our main theorem is the next. The simple inductive proof parallels the one for [99, Theorem I.2.1].

**Theorem 3.2 ([17])** (i)  $L_{\text{gen}}(G) = L_{\text{acc}}(G^d)$ , for any generating cc grammar  $G$ .

(ii)  $L_{\text{acc}}(G) = L_{\text{gen}}(G^d)$ , for any accepting cc grammar  $G$ .

**Proof.** Let  $G = (V_N, V_T, P, \Omega)$  be a cc grammar, generating or accepting. It is shown by induction over  $k$  that  $y_0 \Rightarrow y_1 \Rightarrow \dots \Rightarrow y_k$  is a derivation according to  $G$  if and only if  $y_k \Rightarrow \dots \Rightarrow y_1 \Rightarrow y_0$  is a derivation according to  $G^d$ . By definition of  $G^d$ , this equivalence is true for  $k = 1$ . Let  $y_0 \Rightarrow y_1 \Rightarrow \dots \Rightarrow y_k \Rightarrow y_{k+1}$  according to  $G$  and assume that  $y_k \Rightarrow \dots \Rightarrow y_1 \Rightarrow y_0$  is a derivation according to  $G^d$ . As, by definition,  $y_{k+1} \Rightarrow y_k$  according to  $G_d$ , we get  $y_{k+1} \Rightarrow y_k \Rightarrow \dots \Rightarrow y_1 \Rightarrow y_0$  according to  $G^d$ .

Let  $\omega \in \Omega$  and  $z \in V_T^*$ . If  $G$  is generating, it follows as a particular case that  $\omega \xrightarrow{*}_G z$  if and only if  $z \xrightarrow{*}_{G^d} \omega$ . Hence,  $L_{\text{gen}}(G) = L_{\text{acc}}(G^d)$ . If  $G$  is accepting, then  $z \xrightarrow{*}_G \omega$  if and only if  $\omega \xrightarrow{*}_{G^d} z$ , thus  $L_{\text{acc}}(G) = L_{\text{gen}}(G^d)$ .  $\square$

Since we can interpret any type- $n$  grammar as cc grammar  $G$ , and since we can re-interpret the dual  $G^d$  as type- $n$  grammar, we obtain  $\mathcal{L}^{\text{gen}}(X) = \mathcal{L}^{\text{acc}}(X)$  for  $X \in \{\text{REG}, \text{CF}, \text{CS}, \text{RE}\}$  as a simple corollary. But we can handle other devices in this setting, too. Its application is always the same. (1) Find a characterization of the grammars in question in terms of cc grammars. (2) Consider the duals of the cc grammars in the opposite mode (either accepting if we start with a generating device or vice versa). (3) Re-interpret the dual cc grammars again in terms of the original mechanism. All the following examples are from [17].

**Example 3.3 (Random context grammars)** Every random context grammar  $G$ ,  $G = (V_N, V_T, P, S)$  may be interpreted as a cc grammar  $G' = (V_N, V_T, \{P'\}, \{S\})$  where, for each random context rule  $(\alpha \rightarrow \beta, Q, R)$ , we have one production  $(\alpha \rightarrow \beta, g)$  in  $P'$  such that  $(x_1 \# x_2, n) \in g$  if and only if  $A \in Q$  implies  $A \in \text{Sub}(x_1) \cup \text{Sub}(x_2)$  and  $A \in R$  implies  $A \notin \text{Sub}(x_1) \cup \text{Sub}(x_2)$ , and  $n = 1$ .

Conversely, consider a cc grammar of the form  $G = (V_N, V_T, \{P\}, \{S\})$  with  $S \in V_N$  and productions  $(\alpha \rightarrow \beta, g)$  such that  $\alpha \rightarrow \beta$  is a rewriting rule over  $V_G$  and there are subsets  $Q$  and  $R$  of  $V_N$  such that  $g$  may be characterized via  $(x_1 \# x_2, n) \in g$  if and only if  $A \in Q$  implies  $A \in \text{Sub}(x_1) \cup \text{Sub}(x_2)$  and  $A \in R$  implies  $A \notin \text{Sub}(x_1) \cup \text{Sub}(x_2)$ , and  $n = 1$ . To  $G$ , there corresponds an equivalent random context grammar  $G'$  defined in the obvious way. Hence, we immediately obtain the equivalence between generating and accepting mode for random context grammars.

Other regulation mechanisms that may be treated similarly are:

- string random context grammars (also called semi-conditional grammars) [41, 86],
- random string context grammars ([16, 29]),
- a variant of conditional grammars introduced by Navrátil [82, 85].

**Example 3.4 (Conditional grammars)** To any conditional grammar  $G$  of type  $(i, j)$ ,  $G = (V_N, V_T, P, S, \rho)$ , there corresponds a cc grammar  $G' = (V_N, V_T, \{P'\}, \{S\})$  with  $\alpha \rightarrow \beta \in P$  if and only if  $(\alpha \rightarrow \beta, \text{gsm}_{\alpha \rightarrow \#}(\rho(\alpha \rightarrow \beta)) \times \{1\}) \in P'$ , where the gsm-mapping  $\text{gsm}_{\alpha \rightarrow \#}$  replaces exactly one occurrence of  $\alpha$  by  $\#$ , and keeps the other symbols unchanged. Since type- $j$  languages are closed under non-erasing gsm-mappings,  $\text{gsm}_{\alpha \rightarrow \#}(\rho(\alpha \rightarrow \beta))$  is a type- $j$  language over  $V_G \cup \{\#\}$  with exactly one occurrence of  $\#$ .

On the other hand, if  $G' = (V_N, V_T, \{P'\}, \{S\})$  is a cc grammar with  $S \in V_N$  and productions  $(\alpha \rightarrow \beta, L(\alpha \rightarrow \beta) \times \{1\}) \in P'$  (where  $\alpha \rightarrow \beta$  is a type- $i$  production), then the conditional grammar  $G = (V_N, V_T, P, S, \rho)$  with  $\alpha \rightarrow \beta \in P$  if and only if  $(\alpha \rightarrow \beta, L(\alpha \rightarrow \beta) \times \{1\}) \in P'$ , and  $\rho(\alpha \rightarrow \beta) = \text{gsm}_{\alpha \rightarrow \#}^{-1}(L(\alpha \rightarrow \beta)) \cap V_G^*$ , is equivalent to  $G'$  and is indeed of type  $(i, j)$ , since type- $j$  languages are closed under both inverse gsm-mappings and intersections with regular sets, which means that  $\rho(\alpha \rightarrow \beta)$  is a type- $j$  language over  $V_G$ .

**Example 3.5 (Matrix grammars without appearance checking)** Consider a matrix grammar  $G = (V_N, V_T, M, S, F)$  with  $F = \emptyset$  and let  $m_1, m_2, \dots, m_t$  be the matrices in  $M$  and  $(\alpha_{i1} \rightarrow \beta_{i1}, \alpha_{i2} \rightarrow \beta_{i2}, \dots, \alpha_{ik_i} \rightarrow \beta_{ik_i})$  be the production sequence of matrix  $m_i$ . We introduce a unique label  $p_{ij}$  ( $1 \leq i \leq t, 1 \leq j \leq k_i$ ). In the equivalent cc grammar  $G' = (V'_N, V_T, P, \{S'\})$  with  $V'_N = V_N \cup \{p_{ij} \mid 1 \leq i \leq t, 1 \leq j \leq k_i\} \cup \{S'\}$  (the unions being disjoint), we have a start table containing  $\{(S' \rightarrow Sp_{i1}, \{\#\} \times \{1\}) \mid 1 \leq i \leq t\}$ , a termination table  $\{(p_{i1} \rightarrow \lambda, V_T^* \{\#\} \times \{1\}) \mid 1 \leq i \leq t\}$ , and, for each  $p_{ij}, j < k_i$ , a table  $\{(\alpha_{ij} \rightarrow \beta_{ij}, V_G^* \{\#\} V_G^* \{\#\} \times \{1\}), (p_{ij} \rightarrow p_{i,j+1}, V_G^* \{\#\} V_G^* \{\#\} \times \{2\})\}$ , and, for each  $p_{ik_i}$ , a table  $\{(\alpha_{ik_i} \rightarrow \beta_{ik_i}, V_G^* \{\#\} V_G^* \{\#\} \times \{1\})\} \cup \{(p_{ik_i} \rightarrow p_{s1}, V_G^* \{\#\} V_G^* \{\#\} \times \{2\}) \mid 1 \leq s \leq t\}$ .

On the other hand, a cc grammar of the given form can be readily transformed into an equivalent matrix grammar without appearance checking.

A similar but more awkward construction is possible in the non-erasing case. An alternative proof for this case via a direct construction of dual matrix grammar is presented in [16].

Moreover, both for

- programmed grammars without appearance checking and for
- grammars with regular control without appearance checking [41, 58, 99],

Theorem 3.2 applies in a very similar manner. For the latter case a construction is given in [15].

Now we turn to systems with parallel rewriting. In [52], the concept of accepting parallel derivation is discussed, arriving at the following formal definition:

**Example 3.6 (ETOL systems)** An accepting ETOL system is a quadruple

$$G = (V, V', \{P_1, P_2, \dots, P_r\}, \omega),$$

where  $V'$  is a non-empty subset of the alphabet  $V$ ,  $\omega \in V^+$ , and each table  $P_i$  is a finite subset of  $V^* \times V$  which satisfies the condition that, for each  $a \in V$ , there is a word  $w_a \in V^*$  such that  $(w_a, a) \in P_i$  (the elements of  $P_i$  are written as  $w_a \rightarrow a$  again), such that each  $P_i$  defines a finite substitution  $\sigma_i : V^* \rightarrow 2^{V^*}$ ,  $\sigma_i(a) = \{w \mid w \rightarrow a \in P_i\}$ . We write  $y \Rightarrow x$  if and only if  $x \in \sigma_i^{-1}(y)$  for some  $i$ , and  $L_{\text{acc}}(G) = \{v \in V'^* \mid v \xRightarrow{*} \omega\}$ . As in the generating

case, if no table  $P_i$  contains a  $\lambda$ -rule, that is,  $P_i \subseteq V^+ \times V$  for each  $i$ , then we call the system propagating and add the letter P to the notation of the system. As regards the comparison of the descriptive power of generating and accepting mode, L systems are quite trivial, since any application of a finite substitution in generating mode can be simulated by an application of the inverse of a finite substitution in accepting mode and vice versa. This fact is equally easily seen using cc grammars: For any ETOL-table  $t$ , we introduce in the simulating cc grammar a table  $\bar{t}$  consisting exactly of the productions  $(v \rightarrow w, \{\#\}^* \times \mathbb{N})$  whenever  $v \rightarrow w \in t$ . Obviously, this also applies to the propagating case. Moreover, our theorem allows to carry over results on the synchronization degree (which is determined by the number of tables, see [95]) from generating systems to accepting ones. Especially, the equivalence result is valid for EOL and EPOL systems, where the number of tables is restricted to one.

**Example 3.7 ((Exact) uniformly-limited L systems)** As in the case of ETOL systems, the definitions of accepting  $kulETOL_{ex}$  and  $kulETOL$  systems are derived from the definitions of the generating ones.<sup>3</sup> In [52], a proof of the equivalence between generating and accepting mode is given, stemming from the concept of duality introduced for L systems. Using Theorem 3.2, we associate with each production  $v \rightarrow w$  in a table  $P_i$  of an exact  $kulETOL$  system a production  $(v \rightarrow w, (V_G^* \{\#\})^k V_G^* \times \{1, 2, \dots, k\})$  in a simulating table  $P'_i$  of the corresponding cc grammar.

Analogously, the employment of Theorem 3.2 shows the equivalence between generating and accepting mode of  $kulETOL$  systems.

Note that this theorem allows also to carry over results on the synchronization degree from generating systems to accepting ones.

**Example 3.8 (Scattered context grammars)** The application of Theorem 3.2 is possible as follows. For any production  $(v_{i1}, v_{i2}, \dots, v_{ir_i}) \rightarrow (w_{i1}, w_{i2}, \dots, w_{ir_i})$  of a scattered context grammar, we introduce a table  $t_i$  in the simulating cc grammar consisting of the productions  $(v_{ij} \rightarrow w_{ij}, (V_G^* \{\#\})^{r_i} V_G^* \times \{j\})$  for each  $1 \leq j \leq r_i$ .

Clearly, also in the case of *unordered scattered context grammars*, Theorem 3.2 is applicable. We just have to introduce a new table for each possible permutation.

### 3.1.2 When cc Grammars Do Not Help

We would like to treat matrix grammars with appearance checking as an example, where we cannot apply our main theorem on cc grammars. Let  $G = (V_N, V_T, M, S, F)$  be a matrix grammar with appearance checking.

Observe that we could also include appearance checking features in cc grammars which are not reversible, for example, via a table

$$\{(p_{ij} \rightarrow p_{i,j+1}, (V_G^* \setminus (V_G^* \{\alpha_{ij}\} V_G^*)) \{\#\} \times \{1\})\}$$

if  $p_{ij} \in F$ . Taking the dual grammar, we would exclude  $\beta_{ij}$  instead of  $\alpha_{ij}$ , which gives some feeling why appearance checking really behaves different, as it is shown below. Indeed, we find the following example: let  $G = (\{A\}, \{a\}, \{(A \rightarrow aa, A \rightarrow a)\}, A, F)$  be a generating matrix grammar, where  $F$  contains both  $A \rightarrow aa$  and  $A \rightarrow a$ , then we have  $L_{gen}(G) = \{aa\}$

<sup>3</sup>Note that the definitions of the yield relation of  $kulETOL_{ex}$  and  $kulETOL$  systems as given in Chapter 2 are equally suitable for the generating and accepting modes.

but the language accepted by the dual cc grammar obtained via the proposed construction is  $\{a, aa\}$ .

Permitting appearance checking features in the accepting case allows to test both the containment and the noncontainment of a string in the current sentential form. Thus, it is less surprising that we get the following result.

**Theorem 3.9 ([17])** (i)  $\mathcal{L}^{\text{acc}}(\text{M}, \text{CF}-\lambda, \text{ac}) = \mathcal{L}^{\text{gen}}(\text{CS})$ ,

(ii)  $\mathcal{L}^{\text{acc}}(\text{M}, \text{CF}, \text{ac}) = \mathcal{L}^{\text{gen}}(\text{RE})$ .

**Proof.** We only prove  $\mathcal{L}^{\text{gen}}(\text{CS}) \subseteq \mathcal{L}^{\text{acc}}(\text{M}, \text{CF}-\lambda, \text{ac})$  and  $\mathcal{L}^{\text{gen}}(\text{RE}) \subseteq \mathcal{L}^{\text{acc}}(\text{M}, \text{CF}, \text{ac})$ , using the principle of the proof of Theorem 3.3 in [41]. The converse relations can easily be seen by a construction of a linear bounded automaton and a Turing machine, respectively.

i) First, we find that  $\mathcal{L}^{\text{acc}}(\text{M}, \text{CF}-\lambda, \text{ac})$  is closed under finite union: In the case of two given grammars, make the two sets of nonterminals disjoint, add an additional symbol  $S$  (the new goal symbol) and two productions  $S_1 \rightarrow S$ ,  $S_2 \rightarrow S$  ( $S_1, S_2$  being the goal symbols of the original grammars).

Let  $L \in \mathcal{L}^{\text{gen}}(\text{CS})$ ,  $L \subseteq V^+$ . Obviously,

$$L = \bigcup_{a,b \in V} \{a\}d_a^l(d_b^r(L))\{b\} \cup (L \cap (V \cup V^2)).$$

Since the family  $\mathcal{L}^{\text{acc}}(\text{M}, \text{CF}-\lambda, \text{ac})$  is closed under finite union and contains all finite languages, this identity proves that it is sufficient for the proof of the present assertion to show that  $\{a\}K\{b\} \in \mathcal{L}^{\text{acc}}(\text{M}, \text{CF}-\lambda, \text{ac})$  for  $K \in \mathcal{L}^{\text{gen}}(\text{CS})$ ,  $\lambda \notin M$ .

We consider a generating context-sensitive grammar  $G = (V_N, V_T, P, S)$  without  $\lambda$ -productions in Kuroda normal form generating  $K$ . We construct an accepting matrix grammar  $\overline{G} = (\overline{V}_N, V_T \cup \{a, b\}, \overline{M}, \overline{S}, \overline{F})$  as follows. Let  $\overline{V}_N = V_N \cup \{A, B, \overline{S}, E\} \cup \{Y' \mid Y \in V_N\}$  (the unions being disjoint),  $(\overline{V}_N \cup V_T) \setminus \{Y' \mid Y \in V_N\} = \{C_1, C_2, \dots, C_n\}$ , let  $\overline{M}$  contain the following matrices:

- (a)  $(ASB \rightarrow \overline{S})$ ,
- (b)  $(A \rightarrow E, B \rightarrow E, a \rightarrow A, b \rightarrow B)$ ,
- (c)  $(A \rightarrow A, B \rightarrow B, x \rightarrow X)$  for all context-free rules  $X \rightarrow x \in P$ ,
- (d)  $(A \rightarrow A, B \rightarrow B, Y \rightarrow Y', Z \rightarrow Z'$ ,  
 $Y'C_1 \rightarrow E, Y'C_2 \rightarrow E, \dots, Y'C_n \rightarrow E$ ,  
 $C_1Z' \rightarrow E, C_2Z' \rightarrow E, \dots, C_nZ' \rightarrow E$ ,  
 $Y' \rightarrow X, Z' \rightarrow U)$  for  $XU \rightarrow YZ \in P$ ,

and let  $\overline{F}$  contain exactly all rules with the symbol  $E$  on their right-hand sides. The non-terminal  $E$ , once introduced into the sentential form, cannot be removed anymore; thus it serves as a “trap symbol”. Given a terminal string, only matrix (b) can be applied, replacing exactly one occurrence of  $a$  and of  $b$  with the new nonterminals  $A$  and  $B$ , respectively. The rules  $A \rightarrow E$  and  $B \rightarrow E$  guarantee that this matrix can be applied only once. After  $A$  and  $B$  have been introduced, the matrices of types (c) and (d) can be used in order to simulate the productions of  $G$ . There, the rules with the symbol  $E$  on the right-hand sides make sure that the symbols  $Y$  and  $Z$  can be replaced successfully according to some non-context-free



production of  $G$  only if neighboring occurrences of  $Y$  and  $Z$  are primed. Finally, matrix  $A$  can yield the goal symbol if and only if the given terminal string can be generated by  $G$  and  $A$  and  $B$  have been introduced at the left- and rightmost positions, with the help of matrix  $(b)$  in the first step. Therefore,  $L(\overline{G}) = \{a\}K\{b\}$ .

(ii) For any recursively enumerable language  $L \subseteq \Sigma^*$ ,  $\Sigma = \{c_1, c_2, \dots, c_m\}$ , there is a context-sensitive language  $L' \subseteq V^*$ ,  $V \cap \Sigma = \emptyset$ , and a homomorphism  $h$  such that  $L = h(L')$ . Therefore, it is sufficient to add a sequence of productions  $c_1 \rightarrow E$ ,  $c_2 \rightarrow E$ ,  $\dots$ ,  $c_m \rightarrow E$  as the first  $m$  rules in each matrix constructed above, and to introduce matrices  $(w \rightarrow z)$  for  $h(z) = w$ .  $\square$

**Corollary 3.10** ([17]) (i)  $\mathcal{L}^{\text{gen}}(\text{M}, \text{CF}, \text{ac}) = \mathcal{L}^{\text{acc}}(\text{M}, \text{CF}, \text{ac})$ ,

(ii)  $\mathcal{L}^{\text{gen}}(\text{M}, \text{CF}-\lambda, \text{ac}) \subset \mathcal{L}^{\text{acc}}(\text{M}, \text{CF}-\lambda, \text{ac})$ ,

Let us mention that we need the whole power of ( $\lambda$ -free) context-free rules in this construction in the sense that we cannot restrict the accepting matrix grammars to linear or regular core rules in order to get them more powerful than in generating mode. Indeed, for  $X \in \{\text{REG}, \text{LIN}\}$ , we find

$$\mathcal{L}^{\text{acc}}(\text{M}, X, \text{ac}) = \mathcal{L}^{\text{acc}}(\text{M}, X) = \mathcal{L}^{\text{gen}}(X)$$

by the same argument as in generating mode (e.g. see [41]). Hence, for  $X \in \{\text{REG}, \text{LIN}\}$ , we have

$$\mathcal{L}^{\text{acc}}(\text{M}, X, \text{ac}) = \mathcal{L}^{\text{gen}}(\text{M}, X, \text{ac}).$$

Note that the inclusions both  $\mathcal{L}^{\text{gen}}(\text{M}, \text{CF}-\lambda, \text{ac}) \subset \mathcal{L}^{\text{acc}}(\text{M}, \text{CF}-\lambda, \text{ac})$  and  $\mathcal{L}^{\text{gen}}(\text{M}, \text{CF}, \text{ac}) \subset \mathcal{L}^{\text{acc}}(\text{M}, \text{CF}, \text{ac})$  can alternatively be proved by a direct simulation, more precisely, without (additional) nonterminal symbols. In [19], such a proof is given for pure programmed grammars.

In [15, 16], analogous results are obtained for

- programmed grammars with appearance checking,
- regular controlled grammars with appearance checking and
- ordered grammars.

Furthermore, using the same idea, we get this result for 1-limited ETOL systems:

**Theorem 3.11** ([52]) (i)  $\mathcal{L}^{\text{acc}}(1\text{EPTOL}) = \mathcal{L}^{\text{gen}}(\text{CS})$ ,

(ii)  $\mathcal{L}^{\text{acc}}(1\text{ETOL}) = \mathcal{L}^{\text{gen}}(\text{RE})$ .

**Outline of the proof.** The containment of  $\mathcal{L}^{\text{acc}}(1\text{EPTOL})$  in  $\mathcal{L}^{\text{gen}}(\text{CS})$  holds as any accepting 1EPTOL system can be simulated by a linear bounded automaton.

The family  $\mathcal{L}^{\text{acc}}(1\text{EPTOL})$  is closed with respect to finite union and contains all finite languages. Therefore, for the proof of  $\mathcal{L}^{\text{acc}}(1\text{EPTOL}) \supseteq \mathcal{L}^{\text{gen}}(\text{CS})$ , it is sufficient to show that  $\{a\}M\{b\} \in \mathcal{L}^{\text{acc}}(1\text{EPTOL})$  for  $M \in \mathcal{L}^{\text{gen}}(\text{CS})$ ,  $\lambda \notin M$ . Let  $G = (V_N, V_T, P, S)$  be a context-sensitive grammar without  $\lambda$ -productions in Kuroda normal form generating  $M$ . Let us assume a unique label  $r$  being attached to any rule of the form  $XU \rightarrow YZ$  (the set of labels is denoted by  $Lab$ ). We construct a 1EPTOL system  $G' = (V, V', \{\text{init}_a, \text{init}_b, \text{term}\} \cup$

$\{\text{simul-cf}_{X \rightarrow x} \mid X \rightarrow x \in P\} \cup \{\text{simul-cs}_{r,i} \mid r \in \text{Lab}, 1 \leq i \leq 6\}, S', 1)$  accepting  $\{a\}M\{b\}$  as follows. Let

$$V = V_N \cup \{A, B, S', F_l, F_r\} \cup \{(A, r), [A, r], [B, r], (Y, r), [Z, r] \mid r : XU \rightarrow YZ \in P\} \cup V'$$

(the unions being disjoint), where  $V' = V_T \cup \{a, b\}$ .

For brevity, we leave out productions of the form  $F_l \rightarrow x$  which have to be added for right-hand sides  $x$  not present in the table specifications listed above in order to fulfill the completeness restriction for tables inherited from L systems.

1. start/termination/context-free rules:

- (a)  $\text{init}_a = \{a \rightarrow A\} \cup \{x \rightarrow F_r \mid x \in V \setminus V'\}$
- (b)  $\text{init}_b = \{b \rightarrow B\} \cup \{x \rightarrow F_r \mid x \in V \setminus (V' \cup \{A\})\}$
- (c)  $\text{simul-cf}_{X \rightarrow x} = \{x \rightarrow X\} \cup \{(A, r) \rightarrow F_r, [A, r] \rightarrow F_r \mid r \in \text{Lab}\}$  for context-free rules  $X \rightarrow x \in P$ ;
- (d)  $\text{term} = \{ASB \rightarrow S'\} \cup \{x \rightarrow F_r \mid x \in V\}$

2. For each context-sensitive rule  $r : XU \rightarrow YZ \in P$ , we introduce the next tables:

- (a)  $\text{simul-cs}_{r,1} = \{A \rightarrow [A, r], B \rightarrow [B, r]\} \cup \{[A, s] \rightarrow F_r, (A, s) \rightarrow F_r, [B, s] \rightarrow F_r \mid s \in \text{Lab}\}$ ;
- (b)  $\text{simul-cs}_{r,2} = \{Y \rightarrow [Y, r]\} \cup \{A \rightarrow F_r, B \rightarrow F_r, (A, s) \rightarrow F_r, [A, s'] \rightarrow F_r, [B, s'] \rightarrow F_r, [T, s] \rightarrow F_r, (T, s) \rightarrow F_r \mid s \in \text{Lab}, s' \in \text{Lab} \setminus \{r\}, T \in V_N\}$ ;
- (c)  $\text{simul-cs}_{r,3} = \{Z \rightarrow (Z, r)\} \cup \{A \rightarrow F_r, B \rightarrow F_r, (A, s) \rightarrow F_r, [A, s'] \rightarrow F_r, [B, s'] \rightarrow F_r, [T, s'] \rightarrow F_r, (T, s) \rightarrow F_r \mid s \in \text{Lab}, s' \in \text{Lab} \setminus \{r\}, T \in V_N\}$ ;
- (d)  $\text{simul-cs}_{r,4} = \{[A, r] \rightarrow (A, r)\} \cup \{A \rightarrow F_r, B \rightarrow F_r, (A, s) \rightarrow F_r, [A, s'] \rightarrow F_r, [B, s'] \rightarrow F_r, [T, s'] \rightarrow F_r, (T, s') \rightarrow F_r, [Y, r]y \rightarrow F_r \mid s \in \text{Lab}, s' \in \text{Lab} \setminus \{r\}, T \in V_N, z \in V \setminus \{[Y, r]\}, y \in V \setminus \{(Z, r)\}\}$ ;
- (e)  $\text{simul-cs}_{r,5} = \{[B, r] \rightarrow B\} \cup \{A \rightarrow F_r, B \rightarrow F_r, (A, s') \rightarrow F_r, [A, s] \rightarrow F_r, [B, r] \rightarrow F_r, [T, s'] \rightarrow F_r, (T, s') \rightarrow F_r, z(Z, r) \rightarrow F_r \mid s \in \text{Lab}, s' \in \text{Lab} \setminus \{r\}, T \in V_N, z \in V \setminus \{[Y, r]\}, y \in V \setminus \{(Z, r)\}\}$ ;
- (f)  $\text{simul-cs}_{r,6} = \{(A, r) \rightarrow A\} \cup \{(Z, r) \rightarrow U\} \cup \{[Y, r] \rightarrow X\} \cup \{A \rightarrow F_r, [A, s] \rightarrow F_r, [B, s] \rightarrow F_r, (A, s') \rightarrow F_r, [T, s'] \rightarrow F_r, (T, s') \rightarrow F_r \mid s \in \text{Lab}, s' \in \text{Lab} \setminus \{r\}, T \in V_N\}$ .

The general idea of this construction is the same as in the proof of Theorem 3.9. The simulation of a non-context-free production  $r : XU \rightarrow YZ \in P$  is more involved. The table  $\text{simul-cs}_{r,1}$  marks the special nonterminals  $A$  and  $B$  with the label  $r$  of the production, selecting the production from  $P$  to be simulated. Next,  $\text{simul-cs}_{r,2}$  is used for selecting an occurrence of  $Y$  by marking it with  $r$ , as well, leading to  $[Y, r]$ . Then, one occurrence of  $Z$  is marked by  $r$  with the help of  $\text{simul-cs}_{r,3}$ . The table  $\text{simul-cs}_{r,4}$  checks that there is no unmarked symbol to the right of  $[Y, r]$ , whereas  $\text{simul-cs}_{r,5}$  checks that there is no unmarked symbol to the left of the marked  $Z$ . Moreover,  $[B, r]$  is renamed to  $B$ , again. Finally, by  $\text{simul-cs}_{r,6}$  the marked symbols  $Y, Z$  and  $A$  are replaced with  $X, U$  and (unmarked)  $A$ , thus finishing the simulation of an application of  $r$  and yielding a sentential form with no marked symbols, again. The renaming of

symbols to their several copies and the productions with  $F_r$  on the right-hand sides are used to control the order in which the tables are applied. Note that, according to Definition 2.10, any production of a table has to be applied (exactly) once to a sentential form, if possible. This allows to test non-occurrences of symbols. This also guarantees that all markings are done with one and the same label  $r$  during one cycle of applications of  $\text{simul-cs}_{r,1}$  through  $\text{simul-cs}_{r,6}$ .

By this construction, the first equality of the Theorem is proved. The second equality is now proved as the corresponding one of Theorem 3.9.  $\square$

**Corollary 3.12** ([17]) (i)  $\mathcal{L}^{\text{gen}}(\text{1EPTOL}) \subset \mathcal{L}^{\text{acc}}(\text{1EPTOL})$ ,  
(ii)  $\mathcal{L}^{\text{gen}}(\text{1ETOL}) \subseteq \mathcal{L}^{\text{acc}}(\text{1ETOL})$ .

Finally, we are going to consider another restriction of ETOL systems,<sup>4</sup> where the nondeterminism in the yield relation is decreased considerably: given a table and a sentential form, the application of the table to the sentential form yields only one successor word.

**Definition 3.2** ([96]) Let  $G = (\Sigma, \Delta, \{P_1, P_2, \dots, P_r\}, \omega)$  be a generating ETOL system as in Definition 2.7. The system  $G$  is called *deterministic* if any table  $P_j$  defines a homomorphism, that is  $|\sigma_j(a)| = |\{w \mid a \rightarrow w \in P_j\}| = 1$ , for all  $a \in \Sigma$  and  $1 \leq j \leq r$ .

In this case, we add a letter D to the notation of the systems, coming to the class EDTOL.

If we treat the notion of determinism as purely syntactical concept, we are led to the following definition in the accepting case.

**Definition 3.3** ([52]) Let  $G = (\Sigma, \Delta, \{P_1, P_2, \dots, P_r\}, \omega)$  be an accepting ETOL system as defined in Example 3.6. The system  $G$  is called *deterministic* if  $|\{a \mid w \rightarrow a \in P_j\}| \leq 1$ , for all  $w \in \Sigma^*$  and  $1 \leq j \leq r$ .

Our intuition is to conceive a concept in which any decomposition of the sentential form into non-overlapping subwords can yield at most one result in a direct derivation step, whenever a deterministic table is applied to it. Note that this is in accordance with deterministic generating Lindenmayer systems (where any direct derivation step yields *exactly* one resulting word). One might alternatively think of defining deterministic accepting Lindenmayer systems by requiring all tables define an inverse homomorphism. In order to take also this idea into consideration, the concept of symmetrically deterministic systems is introduced which requires unambiguity on both sides of the rewriting rules.

**Definition 3.4** ([52]) Let  $G$  be some ETOL system (be it generating or accepting). The system  $G$  is called *symmetrically deterministic* or ESDTOL system if and only if for any table  $P$  and any two productions  $v_1 \rightarrow w_1, v_2 \rightarrow w_2 \in P$ ,  $v_1 = v_2$  is equivalent to  $w_1 = w_2$ .

It turns out that, in the generating case, this concept can be viewed as some sort of normal form.

**Lemma 3.13** ([52])  $\mathcal{L}^{\text{gen}}(\text{EDTOL}) = \mathcal{L}^{\text{gen}}(\text{ESDTOL})$

---

<sup>4</sup>In principle, one may consider deterministic variants of sequentially rewriting grammar formalisms as well. We refrain from considering deterministic sequential grammars here since we have shown in [6, 7] that, in most cases, only singleton sets or the empty language can be generated by deterministic sequential grammars.

**Proof.** Since the inclusion ‘ $\supseteq$ ’ of the claim is trivial, the other direction ‘ $\subseteq$ ’ remains to be shown. Let  $G = (V, V', \{P_1, P_2, \dots, P_r\}, \omega)$  be a generating deterministic ETOL system with  $V = \{a_1, a_2, \dots, a_t\}$ ,  $V' = \{a_{s+1}, a_{s+2}, \dots, a_t\}$ . Let  $F, \Lambda_1, \Lambda_2, \dots, \Lambda_t$  be new symbols. We define a new total alphabet  $\bar{V} = V \cup \{F, \Lambda_1, \Lambda_2, \dots, \Lambda_t\}$ .

For each table  $P$  in  $G$ , define

$$\text{conflict}(P) = \{a_i \rightarrow x \in P \mid (\exists j \neq i)(a_j \rightarrow x \in P)\} \quad \text{and}$$

$$\text{no-conflict}(P) = P \setminus \text{conflict}(P)$$

Define

$$\begin{aligned} SD(P) &= \text{no-conflict}(P) \cup \{a_i \rightarrow \Lambda_i x \mid a_i \rightarrow x \in \text{conflict}(P)\} \\ &\cup \{\Lambda_i \rightarrow F^{i+1} \mid a_i \in V\} \cup \{F \rightarrow F\}, \quad \text{and} \\ T_j &= \{\Lambda_j \rightarrow \lambda\} \cup \{b \rightarrow b \mid b \in \bar{V} \setminus \{\Lambda_j\}\}. \end{aligned}$$

Consider the symmetrically deterministic system

$$\bar{G} = (\bar{V}, V', \{SD(P_1), SD(P_2), \dots, SD(P_r), T_1, T_2, \dots, T_t\}, \omega).$$

Then  $L_{\text{gen}}(G) = L_{\text{gen}}(\bar{G})$  follows immediately.  $\square$

Next, we show that the condition of determinism is no restriction at all regarding the power of accepting ETOL systems.

**Lemma 3.14** ([52])  $\mathcal{L}^{\text{acc}}(\text{ETOL}) = \mathcal{L}^{\text{acc}}(\text{EDTOL})$

**Proof.** We have to show the inclusion  $\subseteq$ . The proof is very much alike the previous one. Let  $G = (V, V', \{P_1, P_2, \dots, P_r\}, \omega)$  be an accepting ETOL system.

For each table  $P$  in  $G$ , define

$$\text{conflict}(P) = \{x \rightarrow y \in P \mid (\exists z \neq y)(x \rightarrow z \in P)\} \quad \text{and}$$

$$\text{no-conflict}(P) = P \setminus \text{conflict}(P)$$

We introduce new symbols  $\Lambda_{x \rightarrow y}$  if there is a table  $P$  such that  $x \rightarrow y \in \text{conflict}(P)$ . We assume a suitable enumeration of these symbols, that is,

$$\{\Lambda_{x \rightarrow y} \mid (\exists 1 \leq j \leq r)(x \rightarrow y \in \text{conflict}(P_j))\} = \{\Lambda_i \mid 1 \leq i \leq I\}.$$

We define a new total alphabet  $\bar{V} = V \cup \{F, \Lambda_1, \Lambda_2, \dots, \Lambda_I\}$ .

Next, we define

$$\begin{aligned} DET(P) &= \text{no-conflict}(P) \cup \{\Lambda_{x \rightarrow y} x \rightarrow y \mid x \rightarrow y \in \text{conflict}(P)\} \\ &\cup \{F^{i+1} \rightarrow \Lambda_i \mid 1 \leq i \leq I\} \cup \{F \rightarrow F\}, \quad \text{and} \\ T_j &= \{\lambda \rightarrow \Lambda_j\} \cup \{b \rightarrow b \mid b \in \bar{V} \setminus \{\Lambda_j\}\}. \end{aligned}$$

Consider the deterministic system

$$\bar{G} = (\bar{V}, V', \{DET(P_1), DET(P_2), \dots, DET(P_r), T_1, T_2, \dots, T_I\}, \omega).$$

By this construction,  $L_{\text{acc}}(G) = L_{\text{acc}}(\overline{G})$ .  $\square$

In other words, any ETOL language  $L$  can be accepted by a deterministic ETOL system. This contrasts the situation found within generating ETOL systems, since there is an ETOL language  $L$  which cannot be generated by any deterministic ETOL system [96]. Since the dual of an accepting ESDTOL system is again an ESDTOL system generating the same language and *vice versa*, we may state the following corollary.

**Corollary 3.15** ([52])  $\mathcal{L}^{\text{acc}}(\text{ESDTOL}) = \mathcal{L}^{\text{gen}}(\text{ESDTOL}) = \mathcal{L}^{\text{gen}}(\text{EDTOL})$   
 $\subset \mathcal{L}^{\text{acc}}(\text{EDTOL}) = \mathcal{L}^{\text{acc}}(\text{ETOL}) = \mathcal{L}^{\text{gen}}(\text{ETOL})$

By simple modifications of the proofs of Lemmata 3.13 and 3.14 the statements of these Lemmata are shown to be valid also for ((exact) uniformly) limited ETOL systems, see [52]. We do not know what happens if we restrict our attention to propagating systems.

### 3.1.3 Cooperating Distributed Grammar Systems

Let  $\Gamma = (V_N, V_T, S, P_1, P_2, \dots, P_n)$  be a CD grammar system with context-free components. The language *accepted* in the mode  $\gamma$ ,  $\gamma \in \{*, t, \text{full}\} \cup \{=k, \leq k, \geq k \mid k \geq 1\}$ , is defined by

$$L_{\text{acc}}(\Gamma, \gamma) = \{w \in V_T^* \mid w \xrightarrow{\gamma}_{i_1} w_1 \xrightarrow{\gamma}_{i_2} w_2 \dots \xrightarrow{\gamma}_{i_m} w_m = S, \\ \text{with } m \geq 1, 1 \leq i_j \leq n, w_j \in V_{\Gamma}^*, 1 \leq j \leq m\},$$

where the components are accepting context-free grammars. The accepting derivation in the full-competence mode is defined as follows. Following [11], a set of production rules  $P$  is said to be *sentential-form-complete* (*sf-complete*, for short) with respect to a word  $w$ ,  $w \neq \lambda$ , if and only if  $w$  has a factorization  $w = z_1 z_2 \dots z_n$  such that, for each  $i$ ,  $1 \leq i \leq n$ , there is a rule  $z_i \rightarrow \beta \in P$ , for some  $\beta$ . Now, an *accepting* derivation in the full-competence mode by the  $i$ th component is defined by

$$x \xrightarrow{\text{full}}_i y, \text{ iff } x = x_0 \Rightarrow_i x_1 \Rightarrow_i x_2 \Rightarrow_i \dots \Rightarrow_i x_m = y,$$

where for all  $j$ ,  $0 \leq j \leq m-1$ ,  $P_i$  is *sf-complete* with respect to  $x_j$ , and it is not *sf-complete* with respect to  $y$  or  $y = S$ .<sup>5</sup>

First, we show that using the modes  $*$ ,  $=k$ ,  $\leq k$  and  $\geq k$  for CD grammar systems,  $k \geq 1$ , we get trivial equivalences between generating and accepting devices.

**Theorem 3.16** ([53]) Let  $X \in \{\text{CF}, \text{CF} - \lambda\}$ . If  $N \in \mathbb{N} \cup \{\infty\}$  and  $\gamma \in \{*\} \cup \{=k, \leq k, \geq k \mid k \geq 1\}$ , then  $\mathcal{L}^{\text{gen}}(\text{CD}_N, X, \gamma) = \mathcal{L}^{\text{acc}}(\text{CD}_N, X, \gamma)$ .

**Outline of the proof.** We restrict ourselves to the  $=k$ -mode. The equality for the other cases can be shown with similar arguments. Thus, let  $\Gamma = (N, T, S, P_1, P_2, \dots, P_n)$  (with  $n \in \mathbb{N}$ ) be a CD grammar system working in  $=k$ -mode for some  $k \geq 1$ .

We construct the dual CD grammar system  $\Gamma^d$  (with accepting context-free productions) as follows. Let  $G^d = (N, T, S, P_1^d, P_2^d, \dots, P_n^d)$ , where  $P_i^d$  is defined as

$$P_i^d = \{\beta \rightarrow \alpha \mid \alpha \rightarrow \beta \in P_i\}$$

<sup>5</sup>Note that, in the generating case, we can reformulate the definition in terms of *sf-completeness* as well. There,  $P_i$  is required to be *sf-complete* with respect to  $h(x_j)$ , for  $1 \leq i \leq m-1$ , but not to  $h(y)$  or  $y \in T^*$ , where  $h$  is the morphism erasing all terminal symbols and mapping all nonterminals identically.

for  $1 \leq i \leq n$ .

Now, it suffices to show by induction that  $\alpha \xrightarrow{=k}_i \beta$  in  $\Gamma$  if and only if  $\beta \xrightarrow{=k}_i \alpha$  in  $G^d$  for some  $1 \leq i \leq n$ . Consider a  $k$ -step derivation in  $\Gamma$  via production set  $P_i$ :

$$\alpha = \alpha_0 \Rightarrow_i \alpha_1 \Rightarrow_i \dots \Rightarrow_i \alpha_k = \beta.$$

Since  $\alpha_{j-1} \Rightarrow_i \alpha_j$  by a context-free derivation using a production in  $P_i$ , we have  $\alpha_j \Rightarrow_i \alpha_{j-1}$  using the dual production from  $P_i^d$ . The necessity can be seen similarly.  $\square$

Combining this theorem with the results about the generating case, we immediately get:

**Corollary 3.17 ([53])** 1. If  $\gamma \in \{*, =1, \geq 1\} \cup \{\leq k \mid k \geq 1\}$ , then

$$\mathcal{L}^{\text{gen}}(\text{CF}) = \mathcal{L}^{\text{acc}}(\text{CD}_\infty, \text{CF}[-\lambda], \gamma).$$

2. If  $\gamma \in \{=k, \geq k \mid k \geq 2\}$ , then, for any  $n \geq 2$ ,

$$\begin{aligned} \mathcal{L}^{\text{gen}}(\text{CF}) &= \mathcal{L}^{\text{acc}}(\text{CD}_1, \text{CF}[-\lambda], \gamma) \\ &\subset \mathcal{L}^{\text{acc}}(\text{CD}_2, \text{CF}[-\lambda], \gamma) \\ &\subseteq \mathcal{L}^{\text{acc}}(\text{CD}_n, \text{CF}[-\lambda], \gamma) \\ &\subseteq \mathcal{L}^{\text{acc}}(\text{CD}_\infty, \text{CF}[-\lambda], \gamma) \subseteq \mathcal{L}^{\text{gen}}(\text{M}, \text{CF}[-\lambda]). \end{aligned}$$

Next, we turn to the terminating mode. Similarly as in Section 3.1.2, equivalence to context-sensitive grammars will be shown for accepting systems when working in the  $t$ -mode. The following theorem is possibly the strongest one of its kind (in comparison with other characterizations of the context-sensitive languages via accepting regulated devices, confer [16]), since generating CD grammar systems with context-free components working in  $t$ -mode are strictly weaker than ordered grammars regarding their generative capacity. As in general a constructive proof showing  $\mathcal{L}^{\text{gen}}(X) \subseteq \mathcal{L}^{\text{gen}}(Y)$  is transferable into a proof showing  $\mathcal{L}^{\text{acc}}(X) \subseteq \mathcal{L}^{\text{acc}}(Y)$ , it is interesting to find generative devices with low generative capacity whose accepting counterparts have maximal accepting power, that is, they characterize the context-sensitive languages, if  $\lambda$ -rules are forbidden. Here, in contrast to our earlier results, the admittance of erasing productions does not enhance the accepting power of CD grammar systems working in  $t$ -mode:

Having an accepting component with  $\lambda$ -rules, that is, rules of the form  $\lambda \rightarrow A$ , working in  $t$ -mode we can delete the whole component, because a derivation using such a component does not terminate. Thus,  $\mathcal{L}^{\text{acc}}(\text{CD}_\infty, \text{CF}, t)$  and  $\mathcal{L}^{\text{acc}}(\text{CD}_\infty, \text{CF} - \lambda, t)$  denote the same family of languages.

**Theorem 3.18 ([53])**  $\mathcal{L}^{\text{acc}}(\text{CD}_\infty, \text{CF} - \lambda, t) = \mathcal{L}^{\text{acc}}(\text{CD}_\infty, \text{CF}, t) = \mathcal{L}^{\text{gen}}(\text{CS})$ .

**Proof.** By our above remark, we may assume that a given accepting grammar system  $\Gamma$  working in  $t$ -mode does not contain  $\lambda$ -productions. Hence, it is easy to construct a simulating linear bounded automaton accepting  $L_{\text{acc}}(\Gamma)$ . Therefore, the inclusion  $\mathcal{L}^{\text{acc}}(\text{CD}_\infty, \text{CF}, t) \subseteq \mathcal{L}^{\text{gen}}(\text{CS})$  follows. It is left to show the converse inclusion.

By a standard argument, it can be shown that  $\mathcal{L}^{\text{acc}}(\text{CD}_\infty, \text{CF}, t)$  is closed under union and embraces the finite languages.

Let  $L \in \mathcal{L}^{\text{gen}}(\text{CS})$ ,  $L \subseteq V_T^*$ . Then,

$$L = \bigcup_{a,b \in V_T} (\{a\}V_T^+\{b\} \cap L) \cup (L \cap V_T) \cup (L \cap V_T^2).$$

Since  $L$  is context-sensitive,  $L_{ab} = \{w \in V_T^+ \mid awb \in L\}$  is context-sensitive due to the closure of  $\mathcal{L}^{\text{gen}}(\text{CS})$  under derivatives and intersection with regular languages. Now, it is sufficient for the proof of the present assertion to show that  $\{a\}M\{b\} \in \mathcal{L}^{\text{acc}}(\text{CD}_\infty, \text{CF} - \lambda, t)$  provided that  $M \subseteq V_T^+$  is context-sensitive.

Let  $G = (V_N, V_T, S, P)$  be a context-sensitive grammar without  $\lambda$ -productions in Kuroda normal form generating  $M$ . Let us assume a unique label  $r$  being attached to any genuine context-sensitive rule of the form  $XU \rightarrow YZ$  with  $X, U, Y, Z \in V_N$ ; the set of labels is denoted by  $\text{Lab}_{\text{CS}}$  and let  $\text{Lab}_{\text{CS}} = \{r_1, r_2, \dots, r_R\}$ .

We construct a  $(\text{CD}_\infty, \text{CF} - \lambda, t)$  system of grammars

$$G' = (V'_N, V_T, S', P_{\text{init}}, P_{\text{fin}}, P_{\text{cf},1}, P_{\text{cf},2}, P_{1,1}, P_{1,2}, P_{1,3}, \dots, P_{R,1}, P_{R,2}, P_{R,3})$$

accepting  $\{a\}M\{b\}$ . The nonterminal alphabet is

$$\begin{aligned} V'_N = & V_N \cup \{[A, \rho], (A, \rho), [Y, \rho], (Z, \rho) \mid r_\rho : XU \rightarrow YZ \in P\} \cup \\ & \{\tilde{C} \mid C \in V_N \cup V_T\} \cup \{\bar{c} \mid c \in V_T\} \cup \{A, B, S', F, A'\} \end{aligned}$$

(the unions being disjoint). For brevity, we let  $\bar{V}_T = \{\bar{c} \mid c \in V_T\}$  and define the two morphisms  $\hat{\cdot} : (V_T \cup V_N)^* \rightarrow (\bar{V}_T \cup V_N)^*$  by  $\hat{c} = \bar{c}$ , for  $c \in V_T$ , and  $\hat{C} = C$  for  $C \in V_N$ , and, for each production  $XU \rightarrow YZ$  labelled with  $r$ ,  $g_r : (\bar{V}_T \cup V_N)^* \rightarrow (\bar{V}_T \cup V_N \cup \{\tilde{Y}, \tilde{Z}\})^*$  is given by  $g_r(Y) = \tilde{Y}$ ,  $g_r(Z) = \tilde{Z}$ , and  $g_r(x) = x$ , otherwise.

The component

$$\begin{aligned} P_{\text{init}} = & \{a \rightarrow A, b \rightarrow B\} \cup \{c \rightarrow \bar{c} \mid c \in V_T\} \cup \\ & \{Y \rightarrow F \mid Y \in V'_N \setminus (\bar{V}_T \cup \{A, B\})\}, \end{aligned}$$

when applied correctly, turns the left delimiter  $a$  into  $A$  and the right delimiter  $b$  into  $B$ , while the other occurrences of  $a$ 's and  $b$ 's (as well as all the other terminal symbols) are changed into their barred counterparts. The check of the correctness of this application is postponed until the last applicable component

$$P_{\text{fin}} = \{ASB \rightarrow S'\}$$

does its work.

Purely context-free productions are handled by

$$\begin{aligned} P_{\text{cf},1} = & \{\hat{w} \rightarrow \tilde{C} \mid C \rightarrow w \in P\} \cup \{\hat{D} \rightarrow \tilde{D} \mid D \in V_T \cup V_N\} \cup \{A \rightarrow A'\} \cup \\ & \{(A, \rho) \rightarrow F, [A, \rho] \rightarrow F \mid 1 \leq \rho \leq R\} \cup \{a \rightarrow F, b \rightarrow F\}; \end{aligned}$$

$$\begin{aligned} P_{\text{cf},2} = & \{\tilde{D} \rightarrow \hat{D} \mid D \in V_T \cup V_N\} \cup \{A' \rightarrow A\} \cup \\ & \{(A, \rho) \rightarrow F, [A, \rho] \rightarrow F \mid 1 \leq \rho \leq R\} \cup \{a \rightarrow F, b \rightarrow F\}. \end{aligned}$$

First applying  $P_{\text{cf},1}$  and then  $P_{\text{cf},2}$ , we may simulate one application of some context-free rule. Of course,  $P_{\text{cf},1}$  may be used also to simulate a parallel application of various context-free

rules, but this does not matter, since there is a possible trivial sequentialization of these rules corresponding to a valid sequence of derivation steps in  $G$ .

Finally, we introduce three production sets simulating a genuine context-sensitive production  $r_\rho : XU \rightarrow YZ \in P$ :

$$\begin{aligned}
P_{\rho,1} &= \{A \rightarrow [A, \rho], Y \rightarrow [Y, \rho], Y \rightarrow \tilde{Y}, Z \rightarrow (Z, \rho), Z \rightarrow \tilde{Z}\} \cup \\
&\quad \{(A, \sigma) \rightarrow F \mid 1 \leq \sigma \leq R\} \cup \\
&\quad \{[C, \sigma] \rightarrow F \mid 1 \leq \sigma \leq R \wedge \sigma \neq \rho \wedge C \in V_N \cup V_T \cup \{A\}\} \cup \\
&\quad \{a \rightarrow F, b \rightarrow F, A' \rightarrow F\}, \\
P_{\rho,2} &= \{[A, \rho] \rightarrow (A, \rho), A \rightarrow F, A' \rightarrow F, a \rightarrow F, b \rightarrow F, Y \rightarrow F, Z \rightarrow F\} \cup \\
&\quad \{[C, \sigma] \rightarrow F, (C, \sigma) \rightarrow F \mid 1 \leq \sigma \leq R \wedge \sigma \neq \rho \wedge C \in V_N \cup V_T \cup \{A\}\} \cup \\
&\quad \{z(Z, \rho) \rightarrow F \mid z \in V_T \cup V'_N \wedge z \neq [Y, \rho]\} \cup \\
&\quad \{[Y, \rho]y \rightarrow F \mid y \in V_T \cup V'_N \wedge y \neq (Z, \rho)\} \cup \\
&\quad \{\tilde{U} \rightarrow F \mid U \in V_N \cup V_T, U \notin \{Y, Z\}\}, \\
P_{\rho,3} &= \{(A, \rho) \rightarrow A, (Z, \rho) \rightarrow U, [Y, \rho] \rightarrow X, a \rightarrow F, b \rightarrow F, A' \rightarrow F\} \cup \\
&\quad \{[C, \sigma] \rightarrow F, (C, \sigma) \rightarrow F \mid 1 \leq \sigma \leq R \wedge C \in V_N \cup V_T \cup \{A\}\} \cup \\
&\quad \{\tilde{U} \rightarrow F \mid U \in V_N \cup V_T, U \notin \{Y, Z\}\} \cup \{\tilde{Y} \rightarrow Y, \tilde{Z} \rightarrow Z\}.
\end{aligned}$$

Consider the following derivation of  $G$ :

$$S \xRightarrow{*} x_1 X U x_2 \Rightarrow_{r_\rho} x_1 Y Z x_2.$$

In the constructed grammar system, we find the simulation

$$\begin{aligned}
A\hat{x}_1 Y Z \hat{x}_2 B &\xrightarrow{t}_{\rho,1} [A, \rho]g_\rho(\hat{x}_1)[Y, \rho](Z, \rho)g_\rho(\hat{x}_2)B \\
&\xrightarrow{t}_{\rho,2} (A, \rho)g_\rho(\hat{x}_1)[Y, \rho](Z, \rho)g_\rho(\hat{x}_2)B \\
&\xrightarrow{t}_{\rho,3} A\hat{x}_1 X U \hat{x}_2 B
\end{aligned}$$

Finally, we have  $ASB \xrightarrow{t}_{\text{init}} S'$ .

Observe that the second production set only serves for checking whether the first production set has correctly nondeterministically selected two adjacent  $Y$  and  $Z$ . These checks are always possible, since we introduced left and right end-markers  $A$  and  $B$ , respectively. Furthermore, observe that we need both tests,  $z(Z, \rho) \rightarrow F$  and  $[Y, \rho]y \rightarrow F$  in order to prevent shortcuts, i.e., only replacing either  $Y$  by  $[Y, \rho]$  or  $Z$  by  $(Z, \rho)$ .  $\square$

**Corollary 3.19** ([53])  $\mathcal{L}^{\text{gen}}(\text{CD}_\infty, \text{CF}[-\lambda], t) = \mathcal{L}^{\text{gen}}(\text{ET0L}) = \mathcal{L}^{\text{acc}}(\text{ET0L})$   
 $\subset \mathcal{L}^{\text{acc}}(\text{CD}_\infty, \text{CF}[-\lambda], t) = \mathcal{L}^{\text{gen}}(\text{CS}).$

Now, we turn our attention to grammar systems with a limited number of components.

**Theorem 3.20** ([53]) For  $n \geq 2$  we have:

$$\begin{aligned}
\mathcal{L}^{\text{gen}}(\text{CF}) &= \mathcal{L}^{\text{acc}}(\text{CD}_1, \text{CF}[-\lambda], t) \\
&\subset \mathcal{L}^{\text{acc}}(\text{CD}_n, \text{CF}[-\lambda], t) = \mathcal{L}^{\text{acc}}(\text{CD}_\infty, \text{CF}[-\lambda], t) = \mathcal{L}^{\text{gen}}(\text{CS}).
\end{aligned}$$



**Proof.** The relations  $\mathcal{L}^{\text{gen}}(\text{CF}) = \mathcal{L}^{\text{acc}}(\text{CD}_1, \text{CF}[-\lambda], t) \subseteq \mathcal{L}^{\text{acc}}(\text{CD}_2, \text{CF}[-\lambda], t)$  are obvious; the latter inclusion is shown to be strict by Theorem 3.18 and the proof below, where we show the inclusion  $\mathcal{L}^{\text{acc}}(\text{CD}_\infty, \text{CF}[-\lambda], t) \subseteq \mathcal{L}^{\text{acc}}(\text{CD}_2, \text{CF}[-\lambda], t)$ .

Let  $\Gamma = (V_N, V_T, S, P_1, P_2, \dots, P_n)$  a CD grammar system with at least two components. We construct a CD grammar system with two components

$$\Gamma' = (V'_N, V_T, S_1, P'_1, P'_2)$$

accepting  $L_{\text{acc}}(\Gamma, t)$ . The nonterminal alphabet is

$$V'_N = \{A_i \mid A \in (V_N \cup V_T), 1 \leq i \leq 2n\} \cup \{F\}$$

(the unions being disjoint). For  $1 \leq i \leq 2n$ , we now define a morphism  $h_i : (V_N \cup V_T)^* \rightarrow V'^*_N$ ,  $h_i(C) = C_i$ , if  $C \in V_N \cup V_T$ . The components  $P'_1$  and  $P'_2$  are given as follows:

$$\begin{aligned} P'_1 = & \{a \rightarrow a_i \mid a \in V_T \wedge 1 \leq i \leq n\} \cup \\ & \{A_i \rightarrow A_{n+j} \mid A_i, A_{n+j} \in V'_N \wedge 1 \leq i, j \leq n\}, \end{aligned}$$

$$\begin{aligned} P'_2 = & \{A_{n+i} \rightarrow A_i \mid A_{n+i}, A_i \in V'_N \wedge 1 \leq i \leq n\} \cup \\ & \{h_i(\alpha) \rightarrow h_i(A) \mid \alpha \rightarrow A \in P_i \wedge 1 \leq i \leq n\} \cup \\ & \{A_i A_j \rightarrow F \mid A_i, A_j \in V'_N \wedge 1 \leq i, j \leq n \wedge i \neq j\}. \end{aligned}$$

A possible derivation in  $\Gamma'$  has to start by using component  $P'_1$ . From now on, whenever we simulate one step of the original grammar  $\Gamma$  using production set  $P_i$  we have to perform one application of  $P'_1$  and  $P'_2$  in sequence. Here,  $P'_1$  is used to change the indices of the symbols in the appropriate way. Then with  $P'_2$  one applies the productions of the original set  $P_i$ . In addition,  $P'_2$  is used to check whether the first production set has changed the indices correctly, that is, there is no mixture of letters indexed by different  $i, j$ . Consider the following derivation of  $\Gamma$ :

$$w \xrightarrow{t}_{i_1} \alpha_1 \xrightarrow{t}_{i_2} \dots \xrightarrow{t}_{i_{m-1}} \alpha_{m-1} \xrightarrow{t}_{i_m} \alpha_m = S,$$

with  $w \in V'^*_T$ ,  $m \geq 1$ ,  $1 \leq i_j \leq n$ , and  $1 \leq j \leq m$ . In the constructed grammar system, we find the simulation

$$\begin{aligned} w & \xrightarrow{t}_1 h_{n+i_1}(w) \xrightarrow{t}_2 h_{i_1}(\alpha_1) \\ & \xrightarrow{t}_1 h_{n+i_2}(\alpha_1) \xrightarrow{t}_2 h_{i_2}(\alpha_2) \\ & \dots \\ & \xrightarrow{t}_1 h_{n+i_m}(\alpha_{m-1}) \xrightarrow{t}_2 h_{i_m}(\alpha_m) = S_{i_m} \\ & \xrightarrow{t}_1 S_{n+1} \xrightarrow{t}_2 S_1 \end{aligned}$$

□

Comparing the power of generating and accepting CD grammar systems, we obtain:

**Corollary 3.21 ([53])** 1.  $\mathcal{L}^{\text{gen}}(\text{CF}) = \mathcal{L}^{\text{gen}}(\text{CD}_1, \text{CF}[-\lambda], t) = \mathcal{L}^{\text{acc}}(\text{CD}_1, \text{CF}[-\lambda], t)$ .

2.  $\mathcal{L}^{\text{gen}}(\text{CD}_n, \text{CF}[-\lambda], t) \subset \mathcal{L}^{\text{acc}}(\text{CD}_n, \text{CF}[-\lambda], t)$  for  $n \geq 2$ . □

As to the full competence mode of derivation, we find the following result.

**Theorem 3.22** ([11]) (i)  $\mathcal{L}^{\text{acc}}(\text{CD}_{\infty}, \text{CF}-\lambda, \text{full}) = \mathcal{L}^{\text{gen}}(\text{CS})$

(ii)  $\mathcal{L}^{\text{acc}}(\text{CD}_{\infty}, \text{CF}, \text{full}) = \mathcal{L}^{\text{gen}}(\text{RE})$

**Proof.** (i) Since any CD grammar system from  $(\text{CD}_{\infty}, \text{CF}-\lambda, \text{full})$  can be simulated by a linear-bounded automaton (that is, by a context-sensitive grammar), we only show that the reverse inclusion holds, using a modification of the proofs of analogous statements for other types of devices, again. Let us have a generating context-sensitive grammar  $G = (N, T, P, S)$  in Kuroda normal form, without  $\lambda$  productions. Assume that a unique label  $r$  is attached to any context-sensitive rule, of the form  $XU \rightarrow YZ$  with  $X, U, Y, Z \in N$ , in  $P$ . Let us denote the set of labels by  $\text{Lab}(P) = \{r_1, r_2, \dots, r_R\}$ . Let  $\bar{T} = \{\bar{a} \mid a \in T\}$ ,  $\{\bar{T} = \{\bar{a} \mid a \in T\}$  and let  $h$  be a morphism defined by  $h(a) = A$  for  $A \in N$  and  $h(a) = \bar{a}$  for  $a \in T$ . For a string  $w \in (N \cup T)^*$  let us denote by  $\bar{w} = h(w)$ .

We construct an accepting CD grammar system

$$\Gamma = (N', T, S', P_{\text{init}}, P_{\text{CF}}, P_{1,1}, P_{1,2}, P_{1,3}, P'_{1,1}, P'_{1,2}, P'_{1,3} \dots, P'_{R,1}, P'_{R,2}, P'_{R,3})$$

such that  $L_{\text{acc}}(\Gamma, \text{full}) = L(G)$  holds. Let  $\Gamma$  be defined with

$$N' = N \cup \bar{T} \cup \bar{\bar{T}} \cup \{S', F\} \cup \{[A, r], (A, r) \mid A \in N \text{ and } r \in \text{Lab}(P)\} \\ \cup \{x' \mid x \in N \cup \bar{T}\} \cup \{\langle x, r \rangle \mid x \in N \cup \bar{T} \text{ and } r \in \text{Lab}(P)\}$$

(the unions being disjoint). The components of  $\Gamma$  are constructed as follows:

$$P_{\text{init}} = \{a \rightarrow \bar{a}, \bar{a} \rightarrow \bar{\bar{a}}, \bar{\bar{a}} \rightarrow \bar{a} \mid a \in T\}, \\ P_{\text{CF}} = \{S \rightarrow S'\} \cup \{x \rightarrow x \mid x \in N \cup \bar{T}\} \cup \{\bar{w} \rightarrow C \mid C \rightarrow w \in P\} \cup \\ \{Y \rightarrow [Y, r] \mid r : XU \rightarrow YZ \in P\} \cup \{x' \rightarrow x \mid x \in N \cup \bar{T}\} \cup \\ \{\bar{\bar{a}} \rightarrow \bar{a} \mid a \in T\},$$

and, for  $1 \leq r \leq R$ ,  $r : XU \rightarrow YZ$ :

$$P_{r,1} = \{[Y, r] \rightarrow [Y, r], Z \rightarrow (Z, r)\} \cup \{x \rightarrow x \mid x \in N \cup \bar{T}\} \cup \{x' \rightarrow x \mid x \in N \cup \bar{T}\} \\ P_{r,2} = \{[Y, r](Z, r) \rightarrow F\} \cup \{x \rightarrow x \mid x \in N \cup \bar{T}\} \cup \{x \rightarrow \langle x, r \rangle \mid x \in N \cup \bar{T}\} \\ P_{r,3} = \{[Y, r] \rightarrow X, (Z, r) \rightarrow U\} \cup \{x \rightarrow x \mid x \in N \cup \bar{T}\} \cup \\ \{\langle x, r \rangle \rightarrow x' \mid x \in N \cup \bar{T}\} \\ P'_{r,1} = \{[Y, r] \rightarrow [Y, r], Z \rightarrow Z'\} \\ P'_{r,2} = \{[Y, r]Z' \rightarrow F, [Y, r] \rightarrow Y'\} \\ P'_{r,3} = \{Y' \rightarrow X, Z' \rightarrow U, X \rightarrow X\}$$

Production set  $P_{\text{init}}$  is for starting the derivation process. Obviously, by  $P_{\text{CF}}$  context-free derivation steps of  $G$  are simulated whereas the components  $P_{r,1}, P_{r,2}, P_{r,3}$  and  $P'_{r,1}, P'_{r,2}, P'_{r,3}$  simulate applications done by the rule with label  $r$  after replacing exactly one appearance of symbol  $Y$  in the sentential form by  $[Y, r]$ . The first group of production sets handles the situation when the sentential form is of the form  $u[Y, r]Zv$ , with  $uv \in (N \cup \bar{T})^+$ , and the second is for the situation when the sentential form is  $[Y, r]Z$ . In the first case it is necessary to replace a symbol  $\langle x, r \rangle$  in order to leave  $P_{r,3}$  which can only be introduced by application of  $P_{r,2}$ . But

$P_{r,2}$  can be active only if the symbols  $[Y, r]$  and  $(Z, r)$  are neighboring in the “correct” manner or they do not appear at all. In the latter case, the applications of  $P_{r,2}$  and  $P_{r,3}$  remain without any effect. Shortcuts are impossible since a component must be fully competent when applied. Similarly, it is easy to see that production sets  $P'_{r,1}, P'_{r,2}, P'_{r,3}$  can be successfully applied only if the sentential form is of the form  $[Y, r]Z$ . Hence,  $L_{\text{acc}}(\Gamma, \text{full}) = L_{\text{gen}}(G)$ .

(ii) Without loss of generality we can assume the given type-0 grammar to have only rules as a grammar in Kuroda normal form only having rules of the form  $A \rightarrow \lambda$ , with  $A \in N$ , in addition. Thus, we can use the same construction as in (i) only giving additional rules  $\lambda \rightarrow A$  to component  $P_{\text{CF}}$  if needed. The other direction of the proof can be shown by construction of a Turing machine.  $\square$

In CD grammar systems, all components work according to the same strategy. However, the agents of a problem solving system usually have different capabilities. Therefore, a generalization of CD grammar systems called hybrid CD grammar systems has been investigated (Mitrana [79], Păun [88, 89]).

**Definition 3.5** A *hybrid CD grammar systems* (HCD grammar system) of degree  $n$ , with  $n \geq 1$ , is an  $(n + 3)$ -tuple

$$G = (V_N, V_T, S, (P_1, \gamma_1), (P_2, \gamma_2), \dots, (P_n, \gamma_n)),$$

where  $V_N, V_T, S, P_1, P_2, \dots, P_n$  are defined as in CD grammar systems, and, for  $1 \leq i \leq n$ ,  $\gamma_i \in \{*, \text{t}, \text{full}\} \cup \{=k, \leq k, \geq k \mid k \geq 1\}$ . The language *generated* by a HCD grammar system is defined by

$$L_{\text{gen}}(G) := \{w \in T^* \mid S \xrightarrow{\gamma_{i_1}}_{i_1} w_1 \xrightarrow{\gamma_{i_2}}_{i_2} \dots \xrightarrow{\gamma_{i_m}}_{i_m} w_m = w \text{ with} \\ m \geq 1, 1 \leq i_j \leq n, \text{ and } 1 \leq j \leq m \}$$

Analogously, the language  $\mathcal{L}^{\text{acc}}(G)$  *accepted* by a HCD grammar system  $G$  is defined.

The families of languages generated (accepted) by HCD grammar systems with at most  $n$  [ $\lambda$ -free] context-free components are denoted by  $\mathcal{L}^{\text{gen}}(\text{HCD}_n, \text{CF}[-\lambda])$  ( $\mathcal{L}^{\text{acc}}(\text{HCD}_n, \text{CF}[-\lambda])$ , respectively). As above, we write  $\mathcal{L}^{\text{gen}}(\text{HCD}_\infty, \text{CF}[-\lambda])$ ,  $\mathcal{L}^{\text{acc}}(\text{HCD}_\infty, \text{CF}[-\lambda])$ , respectively, if the number of components is not restricted.

In the following, we consider accepting versus generating hybrid CD grammar systems. Since the t-mode is incorporated in such systems, we immediately get from Theorem 3.18 that  $\mathcal{L}^{\text{acc}}(\text{HCD}_\infty, \text{CF}[-\lambda])$  equals  $\mathcal{L}^{\text{gen}}(\text{CS})$ . Moreover, in the presence of  $\lambda$ -productions, HCD grammar systems characterize the recursively enumerable languages, since erasing rules can be simulated in components working in \*-mode. Thus, we get:

**Corollary 3.23** ([53]) 1.  $\mathcal{L}^{\text{acc}}(\text{HCD}_\infty, \text{CF}[-\lambda]) = \mathcal{L}^{\text{gen}}(\text{CS})$ .

2.  $\mathcal{L}^{\text{acc}}(\text{HCD}_\infty, \text{CF}) = \mathcal{L}^{\text{gen}}(\text{RE})$ .  $\square$

Again, we turn our attention to grammar systems with a limited number of grammars in each system. We quote the known facts on generating systems (see Mitrana [79]).

**Theorem 3.24** For  $n \geq 4$ , we have:

$$\begin{aligned} \mathcal{L}^{\text{gen}}(\text{CF}) &= \mathcal{L}^{\text{gen}}(\text{HCD}_1, \text{CF}[-\lambda]) \\ &\subset \mathcal{L}^{\text{gen}}(\text{HCD}_2, \text{CF}[-\lambda]) \\ &\subseteq \mathcal{L}^{\text{gen}}(\text{HCD}_3, \text{CF}[-\lambda]) \\ &\subseteq \mathcal{L}^{\text{gen}}(\text{HCD}_n, \text{CF}[-\lambda]) = \mathcal{L}^{\text{gen}}(\text{HCD}_\infty, \text{CF}[-\lambda]). \end{aligned}$$

The following was proved by Păun [88] (the proof trivially carries over to the case admitting  $\lambda$ -rules).

**Theorem 3.25**  $\mathcal{L}^{\text{gen}}(\text{ETOL}) \subset \mathcal{L}^{\text{gen}}(\text{HCD}_\infty, \text{CF}[-\lambda]) \subseteq \mathcal{L}^{\text{gen}}(\text{M}, \text{CF}[-\lambda], \text{ac})$ .  $\square$

As in the case of CD grammar systems working in t-mode, the situation is also a little bit different in the case of accepting hybrid systems. We show that the hierarchy collapses to the second level.

**Theorem 3.26** ([53]) For  $n \geq 2$ , we have:

$$\begin{aligned} \mathcal{L}^{\text{gen}}(\text{CF}) &= \mathcal{L}^{\text{acc}}(\text{HCD}_1, \text{CF}[-\lambda]) \\ &\subset \mathcal{L}^{\text{acc}}(\text{HCD}_n, \text{CF}[-\lambda]) = \mathcal{L}^{\text{acc}}(\text{HCD}_\infty, \text{CF}[-\lambda]). \end{aligned}$$

**Proof.** The equality  $\mathcal{L}^{\text{gen}}(\text{CF}) = \mathcal{L}^{\text{acc}}(\text{HCD}_1, \text{CF}[-\lambda])$  is immediate, and the strict inclusion  $\mathcal{L}^{\text{acc}}(\text{HCD}_1, \text{CF}[-\lambda]) \subset \mathcal{L}^{\text{acc}}(\text{HCD}_2, \text{CF}[-\lambda])$  was already shown in Theorem 3.20.

The inclusion  $\mathcal{L}^{\text{acc}}(\text{HCD}_\infty, \text{CF}[-\lambda]) \subseteq \mathcal{L}^{\text{acc}}(\text{HCD}_2, \text{CF}[-\lambda])$  can be seen as follows: Mitrană [79, Theorem 4] has shown that, for every generating HCD grammar system, there exists an HCD grammar system generating the same language, having four components working in the t-mode and *one* component working in  $=k$ -mode for some  $k$ . Observe, this proof carries over to the accepting case as well.

Then, we reduce the number of components working in t-mode, using the same construction as in the proof of Theorem 3.20. Here, we construct two components  $P'_1$  and  $P'_2$ , where  $P'_2$  contains the modified rewriting rules of all components working in the t-mode. Now it is easy to see that only component  $P'_2$  has to work in the t-mode. The other component can work in, e.g.,  $*$ -mode as well. Thus, only one t-mode component is left.

Finally, using a standard technique we put together the components working in  $*$ - and  $=k$ -modes. We have to be careful with the last step, because if we introduce new nonterminal symbols, we have to add additional rewriting rules to  $P'_2$ , in order to check whether the constructed set  $P_1$  has been correctly applied. This is rather technical, and the details are omitted here.

This shows that, in the accepting case, two components are sufficient.  $\square$

Comparing the power of generating and accepting HCD grammar systems, we obtain the next results.

**Corollary 3.27** ([53])

1.  $\mathcal{L}^{\text{gen}}(\text{CF}) = \mathcal{L}^{\text{gen}}(\text{HCD}_1, \text{CF}[-\lambda]) = \mathcal{L}^{\text{acc}}(\text{HCD}_1, \text{CF}[-\lambda])$ .
2.  $\mathcal{L}^{\text{gen}}(\text{HCD}_n, \text{CF}[-\lambda]) \subset \mathcal{L}^{\text{acc}}(\text{HCD}_n, \text{CF}[-\lambda])$  for  $n \geq 2$ .
3.  $\mathcal{L}^{\text{gen}}(\text{HCD}_n, \text{CF}) \subseteq \mathcal{L}^{\text{acc}}(\text{HCD}_n, \text{CF})$  for  $n \geq 2$ .

**Proof.** We know that  $\mathcal{L}^{\text{gen}}(\text{M}, \text{CF}[-\lambda], \text{ac})$ , and *a fortiori*  $\mathcal{L}^{\text{gen}}(\text{HCD}_n, \text{CF}[-\lambda])$  is strictly contained in  $\mathcal{L}^{\text{gen}}(\text{CS})$  which equals the class  $\mathcal{L}^{\text{acc}}(\text{HCD}_n, \text{CF}[-\lambda])$  by our above theorem.  $\square$

### 3.1.4 Parallel Communicating Grammar Systems

A *accepting PC grammar system* ([13, 14]) with  $n$  context-free components is a PC grammar system  $\Gamma = (N, K, T, G_1, G_2, \dots, G_n)$  as in Definition 2.14, where the components  $G_i$  are accepting context-free grammars, that is, with sets of rewriting rules  $P_i \subseteq (N \cup K \cup T)^* \times (N \cup K)$ , for  $1 \leq i \leq n$ . The configuration  $(S_1, S_2, \dots, S_n)$  is said to be the goal configuration, whereas an initial configuration is given by a tuple in  $T^* \times (V_\Gamma^*)^{n-1}$ . For two configurations  $(x_1, x_2, \dots, x_n)$  and  $(y_1, y_2, \dots, y_n)$  of  $\Gamma$  the direct derivation step  $(x_1, x_2, \dots, x_n) \Longrightarrow (y_1, y_2, \dots, y_n)$ , if defined as follows: if there is no  $x_i$  which contains any query symbol, then, for  $1 \leq i \leq n$ , either  $x_i \in (N \cup T)^* \setminus \{S_i\}$  and  $x_i \xrightarrow{G_i} y_i$  or  $x_i = y_i = S_i$ ; otherwise, a communication step is performed which is defined as in the generating case. The language accepted by  $\Gamma$  is the set

$$L_{\text{acc}}(\Gamma) = \{x_1 \in T^* \mid (x_1, x_2, \dots, x_n) \xrightarrow{*}(S_1, S_2, \dots, S_n), x_i \in V_\Gamma^*, 2 \leq i \leq n\}.$$

Whereas the *generated* language consists of the terminal strings appearing as sentential forms of the master grammar  $G_1$  in a derivation which started off with the initial configuration  $(S_1, S_2, \dots, S_n)$ , the *accepted* language consists of all terminal words appearing as sentential forms of the master grammar  $G_1$  in the initial configuration of an arbitrary derivation which yields the goal configuration  $(S_1, S_2, \dots, S_n)$ .

The notions of returning, non-returning and centralized systems are inherited from the generating case. For  $X \in \{\text{PC}, \text{CPC}, \text{NPC}, \text{NCPC}\}$  and  $Y \in \{\text{CF}-\lambda, \text{CF}\}$ , the family of languages accepted by a PC grammar system of type  $X$  components of type  $Y$  is denoted by  $\mathcal{L}^{\text{acc}}(X_\infty Y)$ .

First, we show that accepting PC grammar systems with context-free components are as powerful as Turing machines in either mode, even if  $\lambda$ -productions are prohibited.

**Theorem 3.28** ([14]) For  $X \in \{\text{PC}, \text{CPC}, \text{NPC}, \text{NCPC}\}$  and  $Y \in \{\text{CF}-\lambda, \text{CF}\}$ ,

$$\mathcal{L}^{\text{acc}}(X_\infty Y) = \mathcal{L}(\text{RE}).$$

**Proof.** The inclusions  $\mathcal{L}^{\text{acc}}(X_\infty Y) \subseteq \mathcal{L}(\text{RE})$  trivially hold by Turing machine constructions. The converse inclusions are proved by the following simulations of type-0 grammars which are assumed to be given in an appropriate normal form.

Let  $L \in \mathcal{L}(\text{RE})$ . Then there is a type-0 grammar  $G = (V_N, V_T, P, S)$  generating  $L$ , where the set  $P$  of productions can be assumed to contain only rules of the forms  $A \rightarrow BC$ ,  $A \rightarrow a$ ,  $AB \rightarrow CD$ , and  $Z \rightarrow \lambda$ , where  $A, B, C, D \in V_N$ ,  $a \in V_T$ , and  $Z$  is a special nonterminal symbol. This can be seen by combining the idea of the proof of Theorem 9.9 in [99] with the usual construction of Kuroda normal form (see, for example, [71]).

Let the total alphabet  $V_N \cup V_T$  of  $G$  contain  $r$  symbols, say  $V_N \cup V_T = \{x_1, x_2, \dots, x_r\}$ , and let the number of (pairwise different) productions in  $P$  of the form  $AB \rightarrow CD$  be  $n$ . Moreover, let us assume a unique label  $r_i$ ,  $1 \leq i \leq n$ , being attached to each production of this form.

We consider the generalized accepting parallel communicating grammar system

$$\Gamma = (N, K, T, G_1, G_2, \dots, G_{n+2r+1})$$

of degree  $n + 2r + 1$ , where

$$N = V_N \cup \{S_2, S_3, \dots, S_{n+2r+1}\}, K = \{Q_2, Q_3, \dots, Q_{n+2r+1}\} \text{ and } T = V_T$$

where  $S_2, S_3, \dots, S_{n+2r+1}, Q_2, Q_3, \dots, Q_{n+2r+1}$  are additional symbols, and the components are constructed as follows:

$$\begin{aligned} G_1 &= (N \cup K, T, P_1, S) \text{ with} \\ P_1 &= \{BC \rightarrow A \mid A \rightarrow BC \in P, A, B, C \in V_N\} \\ &\quad \cup \{a \rightarrow A \mid A \rightarrow a \in P, A \in V_N, a \in V_T\} \\ &\quad \cup \{CD \rightarrow Q_{i+1} \mid r_i : AB \rightarrow CD \in P, 1 \leq i \leq n\} \\ &\quad \cup \{x_j \rightarrow Q_{n+1+j} \mid 1 \leq j \leq r\} \cup \{x_j \rightarrow Q_{n+1+r+j} \mid 1 \leq j \leq r\} \end{aligned}$$

and, for  $2 \leq i \leq n+1$ , if  $r_i : AB \rightarrow CD$ ,

$$\begin{aligned} G_i &= (N \cup K, T, P_i, S_i) \text{ with} \\ P_i &= \{A \rightarrow A, AB \rightarrow S_i\}. \end{aligned}$$

Furthermore, for any symbol  $x_j \in V_N \cup V_T$ ,  $1 \leq j \leq r$ , two additional components  $G_{n+1+j}$  and  $G_{n+1+r+j}$  are introduced providing the strings  $Zx_j$  and  $x_jZ$ , respectively. More precisely, for  $1 \leq j \leq r$ , we have

$$\begin{aligned} G_{n+1+j} &= (N \cup K, T, P_{n+1+j}, S_{n+1+j}) \text{ with} \\ P_{n+1+j} &= \{Z \rightarrow Z, Zx_j \rightarrow S_{n+1+j}\}, \\ G_{n+1+r+j} &= (N \cup K, T, P_{n+1+r+j}, S_{n+1+r+j}) \text{ with} \\ P_{n+1+r+j} &= \{Z \rightarrow Z, x_jZ \rightarrow S_{n+1+r+j}\}. \end{aligned}$$

Obviously, a derivation with initial configuration  $(w, \alpha_2, \alpha_3, \dots, \alpha_{n+2r+1})$  yields the goal configuration if

- $S \xRightarrow{*} w$  in  $G$ ,
- for  $2 \leq i \leq n+1$ ,  $\alpha_i = AB$  if  $AB$  is the left-hand side of production  $r_i$  in  $P$ , and
- for  $1 \leq j \leq r$ ,  $\alpha_{n+1+j} = Zx_j$  and  $\alpha_{n+1+r+j} = x_jZ$ .

Then the master component can directly simulate the (reverse) application of context-free productions from  $P$  whereas the “real” monotone productions as well as the  $\lambda$ -productions are simulated by communication steps. Clearly, in those accepting derivations, the components  $G_i$ ,  $i \geq 2$ , must behave such that the  $\alpha_i$ 's are simultaneously rewritten by  $S_i$  exactly in the tact when the master grammar derives its axiom  $S$ . Otherwise, the derivation might be blocked, since there is no component which can rewrite any  $S_i$ ,  $i \geq 2$ , but an application of a rule  $\beta \rightarrow S_i$  at a “wrong moment” does not allow the master to derive words which are not in  $L_{\text{gen}}(G)$ .

Hence,  $L_{\text{acc}}(\Gamma) = L_{\text{gen}}(G) = L$  for the (centralized) PC grammar system  $\Gamma$  both in returning and in non-returning mode. Note that all productions occurring in a component of  $\Gamma$  are accepting context-free and that no  $\lambda$ -rules are needed.  $\square$

Unfortunately, by this construction, both the number of nonterminals and the number of components in the simulating PC grammar system depend on the size of the type-0 grammar to be simulated. We do not know whether or not any given type-0 grammar can be simulated by a PC grammar system with a bounded number of nonterminals and/or components. In case of generating PC grammar systems such bounds are known, see, for instance, [33].

Note that in case of both generating and accepting PC grammar systems, query symbols can be introduced in some sentential form only by rewriting steps and they can be replaced only by means of communication. Hence, we can assume without loss of generality that in the generating case query symbols never appear on left-hand sides of productions of the components. If we take this restricted definition of a generating PC grammar system then we obtain a quite different derivational ability in accepting mode. Formally, this leads to the following definition.

**Definition 3.6** ([14]) A restricted generating [accepting] parallel communicating grammar system is a construct  $\Gamma = (N, K, T, G_1, G_2, \dots, G_n)$  as in Definition 2.14, but  $P_i \subseteq N \times (N \cup K \cup T)^*$  [ $P_i \subseteq (N \cup K \cup T)^* \times N$ ] holds for each component  $G_i = (N \cup K, T, P_i, S_i)$ ,  $1 \leq i \leq n$ .

The languages  $L_{\text{gen}}(\Gamma)$  and  $L_{\text{acc}}(\Gamma)$  generated [accepted] by restricted PC grammar systems are defined as in the non-restricted cases.

For  $X \in \{\text{PC}, \text{CPC}, \text{NPC}, \text{NCPC}\}$  and  $Y \in \{\text{CF}-\lambda, \text{CF}\}$ , we denote the family of languages generated by generalized generating [accepting] PC grammar systems by  $\mathcal{L}^{\text{gen}}(\text{r}X_{\infty}Y)$  [ $\mathcal{L}^{\text{acc}}(\text{r}X_{\infty}Y)$ ].

Obviously, the generative power of generating parallel communicating grammar systems is not altered if we limit to the restricted version:

**Lemma 3.29** ([14]) For  $X \in \{\text{PC}, \text{CPC}, \text{NPC}, \text{NCPC}\}$  and  $Y \in \{\text{CF}-\lambda, \text{CF}\}$ ,

$$\mathcal{L}^{\text{gen}}(X_{\infty}Y) = \mathcal{L}^{\text{gen}}(\text{r}X_{\infty}Y).$$

However, in the case of accepting PC grammar systems the limitation to restricted versions considerably decreases the power as shown in the following lemma.

**Lemma 3.30** ([14]) For  $X \in \{\text{PC}, \text{CPC}, \text{NPC}, \text{NCPC}\}$  and  $Y \in \{\text{CF}-\lambda, \text{CF}\}$ ,

$$\mathcal{L}^{\text{acc}}(\text{r}X_{\infty}Y) = \mathcal{L}(\text{CF}).$$

**Proof.** Let  $\Gamma = (N, K, T, G_1, G_2, \dots, G_n)$  be an accepting PC grammar system with the components  $G_i = (N \cup K, T, P_i, S_i)$ ,  $1 \leq i \leq n$ . By definition,  $P_i \subseteq (N \cup K \cup T)^* \times N$ ,  $1 \leq i \leq n$ . By the priority of communication steps over rewriting steps, we can assume that  $P_i \subseteq (N \cup T)^* \times N$ . This implies that rewriting steps rewrite words over  $N \cup T$  to words over  $N \cup T$ . Therefore a communication step can only occur at the first step. Since in the initial configuration the first component  $x_1$  is a terminal word, it is not changed by a communication step. Thus the language generated by the accepting PC grammar system  $\Gamma$  coincides with the language generated (or, equivalently, accepted) by the context-free grammar  $G_1$ . Hence  $L_{\text{acc}}(\Gamma) \in \mathcal{L}(\text{CF})$  and thus  $\mathcal{L}^{\text{acc}}(\text{PC}_{\infty}\text{CF}) \subseteq \mathcal{L}(\text{CF})$ .

On the other hand, a context-free grammar  $G = (N, T, P, S)$  can be considered as a PC grammar system  $\Gamma = (N, \emptyset, T, G)$ . Obviously,  $L_{\text{acc}}(\Gamma) = L_{\text{acc}}(G)$  (and  $L_{\text{acc}}(G) = L_{\text{gen}}(G)$ ) which implies that  $\mathcal{L}(\text{CF}) \subseteq \mathcal{L}^{\text{acc}}(\text{PC}_{\infty}\text{CF})$ .  $\square$

In conclusion, we list the following relationships between generating and accepting PC grammar systems which are obtained by the results given in this section and in Chapter 2.

**Corollary 3.31** ([14]) Let  $X \in \{\text{CF}, \text{CF}-\lambda\}$  and  $Y \in \{\text{N}, \lambda\}$ . The following relations hold:

$$(i) \quad \mathcal{L}^{\text{acc}}(\text{r}Y\text{PC}_{\infty}X) \subset \mathcal{L}^{\text{gen}}(Y\text{PC}_{\infty}X) = \mathcal{L}^{\text{acc}}(Y\text{PC}_{\infty}X).$$

$$(ii) \mathcal{L}^{\text{acc}}(\text{rYCPC}_{\infty}, \text{CF}-\lambda) \subset \mathcal{L}^{\text{gen}}(\text{YCPC}_{\infty}, \text{CF}-\lambda) \subset \mathcal{L}^{\text{acc}}(\text{YCPC}_{\infty}, \text{CF}-\lambda)$$

$$(iii) \mathcal{L}^{\text{acc}}(\text{rYCPC}_{\infty}, \text{CF}) \subset \mathcal{L}^{\text{gen}}(\text{YCPC}_{\infty}, \text{CF}) \subseteq \mathcal{L}^{\text{acc}}(\text{YCPC}_{\infty}, \text{CF}) \quad \square$$

In fact, Lemma 3.30 argues that there is no reason to consider restricted PC grammar systems (as far, as in our setting, language families are concerned) since they do not yield new language families both in the generating and accepting case. Therefore, in the following we restrict ourselves to the non-restricted variant.

In what follows we deal with another interpretation of analyzing derivations. We look for a definition of analyzing PC grammar systems in such a way, that analyzing derivations mimic their generative counterparts performing the same derivation steps backwards. Our goal is to use exactly the same system in both ways, to generate and to analyze a language. The generating and accepting versions of a parallel communicating grammar system as considered so far in this section do not satisfy this requirement as one can see from the results and proofs above. In order to distinguish the grammars considered now from those considered so far we call them *analyzing PC grammar systems*.

We mention a problem arising from the treatment of terminal strings in generating PC grammar systems. If a component generates a terminal string, this string remains unchanged through the rest of the derivation. Thus, in the analyzing derivation a terminal string can remain unchanged simulating a generating derivation step on a terminal string or it can be changed simulating a generating derivation step backwards. This is artificial since by the productions of the component a change is possible in any moment of time.

In order to eliminate this feature from analyzing derivations we have to eliminate the maintenance of terminal strings from the generating derivations. Thus we make a slight modification in defining derivation steps in the generative mode, a modification which will enable us to find analyzing counterparts to each generative derivation and vice versa. Therefore the equivalence of the generated and accepted language classes will be obvious. After this, we show that the modification of the generating derivation step does not effect the power of returning PC grammar systems in the generative case, so analyzing grammar systems defined this way accept the same class of languages that is generated in the conventional returning generating mode.

Let us start with defining the modified derivation step for the generative mode.

**Definition 3.7** ([14]) Let  $\Gamma = (N, K, T, G_1, G_2, \dots, G_n)$ , for some  $n \geq 1$ , be a generating PC grammar system with initial configuration  $(S_1, S_2, \dots, S_n)$  and let  $(x_1, x_2, \dots, x_n)$  and  $(y_1, y_2, \dots, y_n)$  be two configurations of  $\Gamma$ .  $(x_1, x_2, \dots, x_n)$  directly derives  $(y_1, y_2, \dots, y_n)$  in strong-returning mode, denoted by  $(x_1, x_2, \dots, x_n) \xrightarrow{sr} (y_1, y_2, \dots, y_n)$ , if one of the following three cases holds:

1. There is no  $x_i$  which contains any query symbol, and there is no  $x_i$  which is a terminal word, that is,  $x_i \in (N \cup T)^* \setminus T^*$  for  $1 \leq i \leq n$ . Then  $x_i \xrightarrow{G_i} y_i$ .

2. There is no  $x_i$  which contains any query symbol, that is,  $x_i \in (N \cup T)^*$ ,  $1 \leq i \leq n$ . Then  $y_j = S_j$  if  $x_j \in T^*$ , and  $y_j = x_j$  if  $x_j \in (N \cup T)^* \setminus T^*$ .

3. There is some  $x_i$ ,  $1 \leq i \leq n$ , which contains at least one occurrence of query symbols, that is  $x_i = z_1 Q_{i_1} z_2 Q_{i_2} \dots z_t Q_{i_t} z_{t+1}$  where  $z_j \in (N \cup T)^*$ ,  $1 \leq j \leq t+1$  and  $Q_{i_l} \in K$ ,  $1 \leq l \leq t$ . Then  $y_i = z_1 x_{i_1} z_2 x_{i_2} \dots z_t x_{i_t} z_{t+1}$ , where  $x_{i_l}$ ,  $1 \leq l \leq t$  does not contain any query symbol, and  $y_{i_l} = S_{i_l}$ ,  $1 \leq l \leq t$ . If some  $x_{i_l}$  contains at least one occurrence of query symbols, then  $y_i = x_i$ . For all  $i$ ,  $1 \leq i \leq n$ , for which  $y_i$  is not specified above,  $y_i = x_i$  holds.



The first point is the description of a rewriting step, where no terminal strings are present among the sentential forms. This is a usual rewriting step known from returning PC grammar systems.

The second point is the description of a derivation step, after at least one terminal string appeared among the sentential forms. In this case, the terminal strings are changed to the start symbol, the other ones remain the same.

The third point again is describing a usual returning communication step.

The language generated by systems in strong-returning mode is defined as before, now of course using strong-returning steps during the derivations.

Again, by  $\xrightarrow[sr]{*}$  we denote the reflexive and transitive closure of  $\xrightarrow[sr]{}$

**Definition 3.8** ([14]) Let  $\Gamma = (N, K, T, G_1, G_2, \dots, G_n)$  be a (generating) PC grammar system with master grammar  $G_1$ , and let  $(S_1, S_2, \dots, S_n)$  denote the initial configuration of  $\Gamma$ . The language generated by the PC grammar system  $\Gamma$  in strong-returning mode is

$$L_{sr}(\Gamma) = \{x_1 \in T^* \mid (S_1, S_2, \dots, S_n) \xrightarrow[sr]{*} (x_1, x_2, \dots, x_n), x_i \in V_{\Gamma}^*, 2 \leq i \leq n\}.$$

Let the class of languages generated by PC grammar systems in the strong-returning mode with context-free components be denoted by  $\mathcal{L}^{\text{gen}}(\text{PC}_{\infty}\text{CF}, \text{sr})$ .

Now we show that strong-returning PC grammar systems generate the same class of languages as in the returning mode. To achieve our goal, we make use of earlier results about so called rule-synchronized PC grammar systems. Let us first recall the necessary definitions from [87] and [107].

**Definition 3.9** A transition of a generating parallel communicating grammar system  $\Gamma = (N, K, T, G_1, G_2, \dots, G_n)$  is an  $n$ -tuple  $t = (p_1, p_2, \dots, p_n)$ , with  $p_i \in P_i \cup \{\#\}$  for  $1 \leq i \leq n$ , where  $P_i$  denotes the rule set of the component  $G_i$  and  $\#$  is an additional symbol. We say that transition  $t = (p_1, p_2, \dots, p_n)$  is applied in the rewriting step  $(x_1, x_2, \dots, x_n) \Longrightarrow (y_1, y_2, \dots, y_n)$  of  $\Gamma$ , if  $p_i \in P_i$  implies  $x_i \xrightarrow[p_i]{}$   $y_i$  and  $p_i = \#$  implies  $x_i \in T^*$  and  $y_i = x_i$ .

**Definition 3.10** A PC grammar system with rule-synchronization is an  $(n + 4)$ -tuple  $\Gamma = (N, K, T, G_1, G_2, \dots, G_n, R)$ , where  $(N, K, T, G_1, G_2, \dots, G_n)$  is a usual PC grammar system and  $R \subseteq (P_1 \cup \{\#\} \times P_2 \cup \{\#\} \times \dots \times P_n \cup \{\#\})$  is a set of transitions of  $\Gamma$ . In each rewriting step  $(x_1, x_2, \dots, x_n) \Longrightarrow (y_1, y_2, \dots, y_n)$  of a rule-synchronized PC grammar system one of the transitions from  $R$  must be applied. Communications are performed as usual.

Let  $\mathcal{L}^{\text{gen}}(\text{RPC}_{\infty}\text{CF})$  and  $\mathcal{L}^{\text{gen}}(\text{RPC}_{\infty}\text{CF}, \text{sr})$  denote the families of languages generated by context-free rule-synchronized PC grammar systems in returning and strong returning modes, respectively.

Note that in the strong-returning mode rule-synchronized PC grammar systems never use transitions containing  $\#$ , the “empty rule”, applied to terminal strings in conventional returning systems.

Now we recall a theorem from [34], which shows that rule-synchronization does not add to the generative power of context-free PC grammar systems in the returning and strong-returning modes. There the statement is proved only for returning PC grammar systems, but the same proof also works in the strong-returning case without any modifications.

**Theorem 3.32** (i)  $\mathcal{L}^{\text{gen}}(\text{PC}_{\infty}\text{CF}) = \mathcal{L}^{\text{gen}}(\text{RPC}_{\infty}\text{CF})$ ,

$$(ii) \mathcal{L}^{\text{gen}}(\text{PC}_\infty\text{CF}, \text{sr}) = \mathcal{L}^{\text{gen}}(\text{RPC}_\infty\text{CF}, \text{sr}).$$

Now, with the aid of rule-synchronization we show, through the next two theorems, that context-free systems generate the same class of languages in returning and strong-returning mode.

**Lemma 3.33 ([14])**  $\mathcal{L}^{\text{gen}}(\text{PC}_\infty\text{CF}) \subseteq \mathcal{L}^{\text{gen}}(\text{PC}_\infty\text{CF}, \text{sr})$ .

**Proof.** Let  $\Gamma = (N, K, T, G_1, G_2, \dots, G_n)$  be a context-free returning PC grammar system as above, and let  $R = (P_1 \cup \{\#\} \times P_2 \cup \{\#\} \times \dots \times P_n \cup \{\#\})$  be the set of all the transitions of  $\Gamma$ . Let

$$\Gamma' = (N', K', T, G_1^{(1)}, G_2^{(1)}, \dots, G_n^{(1)}, G_1^{(2)}, G_2^{(2)}, \dots, G_n^{(2)}, G_1^{a_1}, G_2^{a_1}, \dots, G_n^{a_1}, G^{a_2}, R')$$

be a PC grammar system with the nonterminal alphabet  $N'$ , the set  $K'$  of query symbols, the terminal alphabet  $T$  and components  $G_i^{(1)} = (N' \cup K', T, P_i^{(1)}, S)$ ,  $G_i^{(2)} = (N' \cup K', T, P_i^{(2)}, S_i)$ ,  $G_i^{a_1} = (N' \cup K', T, P_i^{a_1}, S)$ ,  $1 \leq i \leq n$ , and  $G^{a_2} = (N' \cup K', T, P^{a_2}, S)$ , where  $G_1^{(2)}$  is the master grammar, and

$$\begin{aligned} N' &= N \cup \{S, \$, Z\} \cup \{[k, j], [k, 2, t], [k, 3, t] \mid 1 \leq k \leq |R|, 1 \leq j \leq 3\}, \\ K' &= \{Q_i^{(j)} \mid 1 \leq i \leq n, 1 \leq j \leq 2\}. \end{aligned}$$

Let also  $h : (N \cup K \cup T)^* \rightarrow (N \cup K' \cup T)^*$  be a homomorphism defined by  $h(x) = x$  for  $x \in (N \cup T)$ ,  $h(Q_j) = Q_j^{(1)}$  for  $Q_j \in K$ , and

$$\begin{aligned} P_i^{(1)} &= \{S \rightarrow S_i Z, Z \rightarrow Z, S \rightarrow S, S \rightarrow Q_i^{(2)}\} \text{ for } 1 \leq i \leq n, \\ P_i^{(2)} &= \{S_i \rightarrow S_i, S_i \rightarrow S_i Z, S_i \rightarrow Q_i^{(1)}, Z \rightarrow Z, Z \rightarrow \lambda\} \\ &\quad \cup \{X \rightarrow h(\alpha) \mid X \rightarrow \alpha \in P_i\} \text{ for } 1 \leq i \leq n, \\ P_i^{a_1} &= \{S \rightarrow S, S \rightarrow Q_i^{(1)}, Z \rightarrow \lambda\} \text{ for } 1 \leq i \leq n, \\ P^{a_2} &= \{S \rightarrow \$, \$ \rightarrow [k, 1], [k, j] \rightarrow [k, j+1], [k, 3] \rightarrow \$ \mid 1 \leq k \leq |R|, 1 \leq j \leq 2\} \\ &\quad \cup \{[k, 1] \rightarrow [k, 2, t], [k, 2, t] \rightarrow [k, 3, t] \mid 1 \leq k \leq |R|\}. \end{aligned}$$

The set  $R'$  is obtained by substituting each transition  $\bar{r}_k$ ,  $1 \leq k \leq |R|$ , in  $R$  by a set of new transitions that can be applied one after the other (this will be ensured by component  $G^{a_2}$ ).

To sketch the idea of the proof, a configuration  $(x_1, x_2, \dots, x_n)$  of  $\Gamma$  will correspond to  $(x_1 Z, x_2 Z, \dots, x_n Z, S, \dots, S, \$)$  in  $\Gamma'$ , where the symbol  $Z$  is a nonterminal distinct from the elements of  $N$ . This way  $\Gamma'$  is not able to take advantage of its strong-returning mode, since no terminal string appears before the end of any derivation.

One transition of  $\Gamma$  is simulated with four new transitions. First the sentential forms are transferred from  $G_1^{(1)}, G_2^{(1)}, \dots, G_n^{(1)}$  to  $G_1^{(2)}, G_2^{(2)}, \dots, G_n^{(2)}$ , where the effect of the rules of  $\Gamma$  is simulated in three rewriting steps, the rewriting rules are applied to the sentential forms and the  $Z$  nonterminals are erased if necessary, before a sentential form is communicated, in order to avoid the duplication of these  $Z$ -s in the strings. Since the application of the “empty rule”  $\#$  is simulated by the application of  $Z \rightarrow Z$  to a sentential form  $xZ$ , it is also necessary to check, whether  $x$  is really terminal or not. This is done by the components  $G_1^{a_1}, G_2^{a_1}, \dots, G_n^{a_1}$ . The correct order of the application of these substituting transitions is ensured by component  $G^{a_2}$ .

Now we construct the substituting transitions. Let  $\bar{r}_k = (r_{k,1}, r_{k,2}, \dots, r_{k,n})$ ,  $1 \leq k \leq |R|$ , be a transition in  $R$ , and let  $Req(k) \subset \{1, 2, \dots, n\}$ ,  $1 \leq k \leq |R|$  contain the indices of those components which are queried after the execution of transition  $\bar{r}_k$ , in other words  $j \in Req(k)$  if and only if there is a rule  $r_{k,l}$  for some  $l$ ,  $1 \leq l \leq n$ , which introduces a string containing  $Q_j$ , the query symbol requesting the sentential form of the  $j$ -th component. We will need this information, since the  $Z$  nonterminals have to be erased from these sentential forms before the communication. Now let  $\mathcal{R}^{(1)}$  contain the transitions

$$(r_{k,1}^{(1,1)}, r_{k,2}^{(1,1)}, \dots, r_{k,n}^{(1,1)}, r_{k,1}^{(1,2)}, r_{k,2}^{(1,2)}, \dots, r_{k,n}^{(1,2)}, r_{k,1}^{(1,a_1)}, r_{k,2}^{(1,a_1)}, \dots, r_{k,n}^{(1,a_1)}, r_k^{(1,a_2)}),$$

for  $1 \leq k \leq |R|$ , where

$$\begin{aligned} r_{k,i}^{(1,1)} &= Z \rightarrow Z \text{ for } 1 \leq i \leq n, \\ r_{k,i}^{(1,2)} &= S_i \rightarrow Q_i^{(1)} \text{ for } 1 \leq i \leq n, \\ r_{k,i}^{(1,a_1)} &= \begin{cases} S \rightarrow S & \text{if } r_{k,i} \neq \# \\ S \rightarrow Q_i^{(1)} & \text{if } r_{k,i} = \# \end{cases} \text{ for } 1 \leq i \leq n, \\ r_k^{(1,a_2)} &= \$ \rightarrow [k, 1]. \end{aligned}$$

Here the components  $G_1^{(1)}, G_2^{(1)}, \dots, G_n^{(1)}$  leave their sentential forms unchanged, while the components  $G_1^{(2)}, G_2^{(2)}, \dots, G_n^{(2)}$  get ready to receive them. Components  $G_1^{a_1}, G_2^{a_1}, \dots, G_n^{a_1}$  query those components which must contain a terminal string (apart from  $Z$ ) according to the transition  $\bar{r}_k$  being simulated.

Let  $\mathcal{R}^{(2)}$  contain the transitions

$$(r_{k,1}^{(2,1)}, r_{k,2}^{(2,1)}, \dots, r_{k,n}^{(2,1)}, r_{k,1}^{(2,2)}, r_{k,2}^{(2,2)}, \dots, r_{k,n}^{(2,2)}, r_{k,1}^{(2,a_1)}, r_{k,2}^{(2,a_1)}, \dots, r_{k,n}^{(2,a_1)}, r_k^{(2,a_2)}),$$

and

$$(r_{k,1}^{(2,t,1)}, r_{k,2}^{(2,t,1)}, \dots, r_{k,n}^{(2,t,1)}, r_{k,1}^{(2,t,2)}, r_{k,2}^{(2,t,2)}, \dots, r_{k,n}^{(2,t,2)}, r_{k,1}^{(2,t,a_1)}, r_{k,2}^{(2,t,a_1)}, \dots, r_{k,n}^{(2,t,a_1)}, r_k^{(2,t,a_2)}),$$

for  $1 \leq k \leq |R|$ , where

$$\begin{aligned} r_{k,i}^{(2,1)} &= S \rightarrow S \text{ for } 1 \leq i \leq n, \\ r_{k,i}^{(2,2)} &= \begin{cases} Z \rightarrow \lambda & \text{if } i \in Req(k) \text{ and } r_{k,i} \neq \# \\ Z \rightarrow Z & \text{if } i \notin Req(k) \text{ or } r_{k,i} = \# \end{cases} \text{ for } 1 \leq i \leq n, \\ r_{k,i}^{(2,a_1)} &= \begin{cases} S \rightarrow S & \text{if } r_{k,i} \neq \# \\ Z \rightarrow \lambda & \text{if } r_{k,i} = \# \end{cases} \text{ for } 1 \leq i \leq n, \\ r_k^{(2,a_2)} &= [k, 1] \rightarrow [k, 2], \end{aligned}$$

and

$$\begin{aligned} r_{k,i}^{(2,t,1)} &= S \rightarrow S \text{ for } 1 \leq i \leq n, \\ r_{k,i}^{(2,t,2)} &= \begin{cases} Z \rightarrow \lambda & \text{if } r_{k,i} \neq \# \\ Z \rightarrow Z & \text{if } r_{k,i} = \# \end{cases} \text{ for } 1 \leq i \leq n, \\ r_{k,i}^{(2,t,a_1)} &= \begin{cases} S \rightarrow S & \text{if } r_{k,i} \neq \# \\ Z \rightarrow \lambda & \text{if } r_{k,i} = \# \end{cases} \text{ for } 1 \leq i \leq n, \\ r_k^{(2,t,a_2)} &= [k, 1] \rightarrow [k, 2, t]. \end{aligned}$$

If the system uses a transition of the first type the derivation is to be continued, if the second type is used then  $\bar{r}_k$  will be the last simulated transition. Using a first type transition the components  $G_1^{(1)}, G_2^{(1)}, \dots, G_n^{(1)}$  wait for the others, while  $G_1^{(2)}, G_2^{(2)}, \dots, G_n^{(2)}$  get ready to apply the effective rewriting rules by erasing the symbol  $Z$  from those sentential forms which are not terminal and have to be communicated according to the rules of  $\bar{r}_k$ . The components  $G_1^{a_1}, G_2^{a_1}, \dots, G_n^{a_1}$  also erase the  $Z$ -s from their sentential forms, which will result in the blocking of the system, if the rest of these strings are not terminal words. The second type transition does the same, except in components  $G_1^{(2)}, G_2^{(2)}, \dots, G_n^{(2)}$  it erases  $Z$ -s from all sentential forms to which a production other than  $\sharp$  is applied according to  $\bar{r}_k$ , and in component  $G^{a_2}$  it introduces a nonterminal which tells the system to stop after the simulation of  $\bar{r}_k$ .

Let  $\mathcal{R}^{(3)}$  contain the transitions

$$(r_{k,1}^{(3,1)}, r_{k,2}^{(3,1)}, \dots, r_{k,n}^{(3,1)}, r_{k,1}^{(3,2)}, r_{k,2}^{(3,2)}, \dots, r_{k,n}^{(3,2)}, r_{k,1}^{(3,a_1)}, r_{k,2}^{(3,a_1)}, \dots, r_{k,n}^{(3,a_1)}, r_k^{(3,a_2)}),$$

and

$$(r_{k,1}^{(3,t,1)}, r_{k,2}^{(3,t,1)}, \dots, r_{k,n}^{(3,t,1)}, r_{k,1}^{(3,t,2)}, r_{k,2}^{(3,t,2)}, \dots, r_{k,n}^{(3,t,2)}, r_{k,1}^{(3,t,a_1)}, r_{k,2}^{(3,t,a_1)}, \dots, r_{k,n}^{(3,t,a_1)}, r_k^{(3,t,a_2)}),$$

for  $1 \leq k \leq |R|$ , where

$$\begin{aligned} r_{k,i}^{(3,1)} &= S \rightarrow S \text{ for } 1 \leq i \leq n, \\ r_{k,i}^{(3,2)} &= \begin{cases} X \rightarrow h(\alpha) & \text{if } r_{k,i} = X \rightarrow \alpha \\ Z \rightarrow \lambda & \text{if } i \in \text{Req}(k) \text{ and } r_{k,i} = \sharp \\ Z \rightarrow Z & \text{if } i \notin \text{Req}(k) \text{ and } r_{k,i} = \sharp \end{cases} \text{ for } 1 \leq i \leq n, \\ r_{k,i}^{(3,a_1)} &= S \rightarrow S \text{ for } 1 \leq i \leq n, \\ r_k^{(3,a_2)} &= [k, 2] \rightarrow [k, 3], \end{aligned}$$

and

$$\begin{aligned} r_{k,i}^{(3,t,1)} &= S \rightarrow S \text{ for } 1 \leq i \leq n, \\ r_{k,i}^{(3,t,2)} &= \begin{cases} X \rightarrow h(\alpha) & \text{if } r_{k,i} = X \rightarrow \alpha \\ Z \rightarrow \lambda & \text{if } r_{k,i} = \sharp \end{cases} \text{ for } 1 \leq i \leq n, \\ r_{k,i}^{(3,t,a_1)} &= S \rightarrow S \text{ for } 1 \leq i \leq n, \\ r_k^{(3,t,a_2)} &= [k, 2, t] \rightarrow [k, 3, t]. \end{aligned}$$

Using transitions of the first type the components  $G_1^{(2)}, G_2^{(2)}, \dots, G_n^{(2)}$  apply the rules of  $\bar{r}_k$  and erase the  $Z$ -s from those strings of which the rest is a terminal word and are going to be communicated after the application of  $\bar{r}_k$ . Then a communication will follow, the queries introduced by  $\bar{r}_k$  are going to be satisfied. The components  $G_1^{a_1}, G_2^{a_1}, \dots, G_n^{a_1}$  block the derivation, if any of them contains a string different from  $S$ , which is the case, if  $\bar{r}_k$  could not be applied to the sentential forms. Transitions of the second type do the same, but also erase all  $Z$ -s and block the system with  $G^{a_2}$  introducing  $[k, 3, t]$ . If a terminal string is generated in  $G_1^{(a_2)}$ , the master, it is the string generated by the system.

Let  $\mathcal{R}^{(4)}$  contain the transitions

$$(r_{k,1}^{(4,1)}, r_{k,2}^{(4,1)}, \dots, r_{k,n}^{(4,1)}, r_{k,1}^{(4,2)}, r_{k,2}^{(4,2)}, \dots, r_{k,n}^{(4,2)}, r_{k,1}^{(4,a_1)}, r_{k,2}^{(4,a_1)}, \dots, r_{k,n}^{(4,a_1)}, r_k^{(4,a_2)}),$$

for  $1 \leq k \leq |R|$ , where

$$\begin{aligned} r_{k,i}^{(4,1)} &= S \rightarrow Q_i^{(2)} \text{ for } 1 \leq i \leq n, \\ r_{k,i}^{(4,2)} &= \begin{cases} S_i \rightarrow S_i Z & \text{if } i \in \text{Req}(k) \\ Z \rightarrow Z & \text{if } i \notin \text{Req}(k) \end{cases} \text{ for } 1 \leq i \leq n, \\ r_{k,i}^{(4,a_1)} &= S \rightarrow S \text{ for } 1 \leq i \leq n, \\ r_k^{(4,a_2)} &= [k, 3] \rightarrow \$ . \end{aligned}$$

These transitions continue the derivations. The sentential forms are sent back to components  $G_1^{(1)}, G_2^{(1)}, \dots, G_n^{(1)}$ , and the system gets ready to simulate an other transition.

Now if we take a special initializing transition

$$\bar{r}_0 = (r_{0,1}^{(1)}, r_{0,2}^{(1)}, \dots, r_{0,n}^{(1)}, r_{0,1}^{(2)}, r_{0,2}^{(2)}, \dots, r_{0,n}^{(2)}, r_{0,1}^{(a_1)}, r_{0,2}^{(a_1)}, \dots, r_{0,n}^{(a_1)}, r_{0,1}^{(a_2)})$$

where, for  $1 \leq i \leq n$ ,

$$\begin{aligned} r_{0,i}^{(1)} &= S \rightarrow S_i Z, & r_{0,i}^{(2)} &= S_i \rightarrow S_i, \\ r_{0,i}^{(a_1)} &= S \rightarrow S, & r_{0,i}^{(a_2)} &= S \rightarrow \$, \end{aligned}$$

and the set of controlling transitions

$$R' = \{\bar{r}_0\} \cup \mathcal{R}^{(1)} \cup \mathcal{R}^{(2)} \cup \mathcal{R}^{(3)} \cup \mathcal{R}^{(4)}$$

then we have a PC grammar systems simulating the returning system  $\Gamma$  in strong-returning mode.  $\square$

Obviously, by the fact  $\mathcal{L}^{\text{gen}}(\text{PC}_{\infty}\text{CF}) = \mathcal{L}(\text{RE})$  (see [33], [70]), we obtain the inclusion  $\mathcal{L}^{\text{gen}}(\text{PC}_{\infty}\text{CF}, \text{sr}) \subseteq \mathcal{L}^{\text{gen}}(\text{PC}_{\infty}\text{CF})$ . We prove this result by a simulation.

**Lemma 3.34** ([14]) For each context-free strong-returning PC grammar system  $\Gamma$  we can construct a returning PC grammar system  $\Gamma'$  which generates the same language as  $\Gamma$ .

**Proof.** Let  $\Gamma = (N, K, T, G_1, G_2, \dots, G_n)$  be a PC grammar system with  $n$  context-free components  $G_i = (N \cup K, T, P_i, S_i)$ ,  $1 \leq i \leq n$ , working in the strong-returning mode, and let  $R = (P_1 \times P_2 \times \dots \times P_n)$  be the set of all transitions of  $\Gamma$ . Note that the transitions do not contain the symbol  $\sharp$ , since the components always return to their axiom after the appearance of a terminal string before doing any effective rewriting. We are going to construct a rule-synchronized returning PC grammar system  $\Gamma'$ , which generates the same language as  $\Gamma$ , and then our statement follows from Theorem 3.32.

Let

$$\Gamma' = (N', K', T, G_1, G_2, \dots, G_n, G^{a_1}, G^{a_2}, R')$$

be a PC grammar system with the nonterminal alphabet  $N'$ , terminal alphabet  $T$ , set  $K'$  of query symbols, the component grammars  $G_i = (N' \cup K', T, P_i, S_i)$ ,  $1 \leq i \leq n$ ,  $G^{a_j} = (N' \cup K', T, P^{a_j}, S)$ ,  $1 \leq j \leq 2$ , and the set  $R'$  of controlling transitions, where

$$\begin{aligned} N' &= N \cup \{X' \mid X \in N\} \cup \{S, S'\}, \\ K' &= K \cup \{Q^{a_1}\}, \end{aligned}$$

and  $G_1$  is the master grammar. The sets of productions are

$$\begin{aligned} P_i &= \{X \rightarrow X', X' \rightarrow \alpha \mid X \rightarrow \alpha \in P_i\}, \quad 1 \leq i \leq n, \\ P^{a_1} &= \{S \rightarrow S, S \rightarrow \gamma a \mid \gamma \in K^*, |\gamma|_{Q_j} \leq 1, 1 \leq j \leq n, a \in T\}, \\ P^{a_2} &= \{S \rightarrow Q^{a_1} S', S' \rightarrow S\}. \end{aligned}$$

The controlling transition set  $R'$  is constructed by creating sets  $\mathcal{R}_k^{(1)}$  and  $\mathcal{R}_k^{(2)}$  of new transitions for each transition  $\bar{r}_k$ ,  $1 \leq k \leq |R|$ , of  $R$ . These transitions simulate  $\bar{r}_k$  in two rewriting steps. The first  $n$ -tuple of rules of these new transitions will simulate the application of a transition from  $R$ , while the rules of the other components will simulate the strong-returning mode by querying those components of the first  $n$ -tuple,  $G_1, G_2, \dots, G_n$ , which produced a terminal string.

For a transition  $\bar{r}_k = (r_{k,1}, r_{k,2}, \dots, r_{k,n})$ ,  $r_{k,i} \in P_i$ ,  $1 \leq i \leq n$ ,  $1 \leq k \leq |R|$ , of  $R$ , let  $\mathcal{R}_k^{(1)}$  and  $\mathcal{R}_k^{(2)}$  be the sets of transitions

$$(r_{k,1}^{(1)}, r_{(k,2)}^{(1)}, \dots, r_{k,n}^{(1)}, \sharp, S \rightarrow Q^{a_1} S')$$

and

$$(r_{k,1}^{(2)}, r_{(k,2)}^{(2)}, \dots, r_{k,n}^{(2)}, r_k^{a_1}, S' \rightarrow S),$$

respectively, where, for  $1 \leq i \leq n$ ,

$$\begin{aligned} r_{k,i}^{(1)} &= X \rightarrow X', \\ r_{k,i}^{(2)} &= X' \rightarrow \alpha \text{ with } r_{k,i} = X \rightarrow \alpha, \\ r_k^{a_1} &\in \{S \rightarrow \gamma a \mid \gamma \in K^*, |\gamma|_{Q_j} \leq 1, 1 \leq j \leq n, a \in T\}. \end{aligned}$$

Now let us take a special initial transition

$$\bar{r}_0 = (S_1 \rightarrow S'_1, S_2 \rightarrow S'_2, \dots, S_n \rightarrow S'_n, S \rightarrow S, S \rightarrow Q^{a_1} S'),$$

let also

$$\mathcal{R}_k = \mathcal{R}_k^{(1)} \cup \mathcal{R}_k^{(2)}$$

and

$$R' = \{\bar{r}_0\} \cup \left( \bigcup_{k=1}^{|R|} \mathcal{R}_k \right).$$

These transitions do the following: the first subset of  $\mathcal{R}_k$  begin the application of a rule in  $G_1, G_2, \dots, G_n$ , which will be finished by a transition of the second subset. During this finishing step, the component  $G^{a_1}$  can query the components  $G_i$ ,  $1 \leq i \leq n$ , to remove the possibly appearing terminal strings. The system blocks the derivation if either a terminal string is not removed, since the applicable transitions never contain  $\sharp$  as one of the rules  $r_{k,i}^{(j)}$ ,  $1 \leq j \leq 2$ ,  $1 \leq i \leq n$ ,  $1 \leq k \leq |R|$ , or if a string containing nonterminals is removed, since the next transition always applies  $\sharp$  to the string made up of the removed sentential forms.  $\square$

Now we define analyzing derivations by “turning around” strong-returning derivation steps.

**Definition 3.11** ([14]) Let  $\Gamma = (N, K, T, G_1, G_2, \dots, G_n)$  be a (generating) PC grammar system as above with axioms  $S_i$ ,  $1 \leq i \leq n$ , and let  $(x_1, x_2, \dots, x_n)$  and  $(y_1, y_2, \dots, y_n)$  be two configurations of  $\Gamma$ . We say that  $(x_1, x_2, \dots, x_n)$  directly derives  $(y_1, y_2, \dots, y_n)$  in analyzing mode, denoted by  $(x_1, x_2, \dots, x_n) \xrightarrow{ana} (y_1, y_2, \dots, y_n)$ , if one of the following three cases holds:

1. For  $1 \leq i \leq n$ ,  $x_i = z_1 \alpha z_2$  for some  $z_1, z_2 \in (N \cup T)^*$ ,  $\alpha \in (N \cup T \cup K)^*$ ,  $y_i = z_1 X z_2$  and  $X \rightarrow \alpha \in P_i$ .
2. If  $x_i \in (N \cup T)^*$  for  $1 \leq i \leq n$ , then either  $y_i = x_i$  or  $y_i \in T^*$  and  $x_i = S_i$ .
3. Let there be at least one  $j$ ,  $1 \leq j \leq n$ , with  $x_j = S_j$ . For  $1 \leq i \leq n$ , if  $|x_i|_K > 0$ , then  $y_i = x_i$ , and if  $|x_i|_K = 0$ , then either  $y_i = x_i$  or  $y_i = z_1 Q_{i_1} z_2 Q_{i_2} \dots z_t Q_{i_t} z_{t+1}$  for some  $z_l \in (N \cup T)^*$ ,  $1 \leq l \leq t+1$ , and some  $Q_{i_k} \in K$ ,  $1 \leq k \leq t$ , such that the following condition holds: If  $y_i = z_1 Q_{i_1} z_2 Q_{i_2} \dots z_t Q_{i_t} z_{t+1}$  then  $x_i = z_1 y_{i_1} z_2 y_{i_2} \dots z_t y_{i_t} z_{t+1}$  and  $y_{i_k} \in (N \cup T)^*$  and  $x_{i_k} = S_{i_k}$  for  $1 \leq k \leq t$ .

The first point is the description of an analyzing rewriting step, each grammar uses one of its rules “backwards” (therefore analyzing grammars work as accepting grammars).

The second point describes the analyzing counterpart of the strong-returning feature: if an axiom is present at some component, it can be replaced with an arbitrary terminal string while the other sentential forms remain unchanged.

The third point describes a communication step, which is possible to perform if the sentential form of at least one of the components is its axiom. In this case, the other components send subwords of their sentential forms to these components (the ones which have the axiom as their current string), and replace the subword they have sent, with the appropriate query symbol ( $Q_j$  for example, if the subword was sent to component  $G_j$ , for some  $j$ ,  $1 \leq j \leq n$ ). According to the classroom model this can be interpreted as a distribution of subtasks to agents who have finished their assignments and recording the distribution by the corresponding query symbol.

By  $\xrightarrow[ana]{*}$  we denote the reflexive and transitive closure of  $\xrightarrow{ana}$ .

**Definition 3.12** ([14]) Let  $\Gamma = (N, K, T, G_1, G_2, \dots, G_n)$  be a PC grammar system. The language analyzed by the PC grammar system  $\Gamma$  is

$$L_{ana}(\Gamma) = \{x_1 \in T^* \mid (x_1, x_2, \dots, x_n) \xrightarrow[ana]{*} (S_1, S_2, \dots, S_n), x_i \in V_\Gamma^*, 2 \leq i \leq n\}.$$

Let the class of languages analyzed by PC grammar systems with context-free components be denoted by  $\mathcal{L}^{ana}(\text{PC}_\infty \text{CF})$ .

Note the following difference between the generating process (with and without strong return) or accepting process on one side and the analyzing process on the other side. In a generating and accepting derivation the current sentential form determines uniquely whether or not a usual derivation step, a derivation step with strong return or a communication step has to be done. In an analyzing derivation we have to make a choice what type of step we want to make backwards. By our motivation we cannot avoid to choose a derivation step or a communication step. On the other hand, if we restrict to parallel communicating grammar systems where, for any production, the axiom does not occur in the word on the right-hand side, then the generative power of generating systems (with and without strong return) is not changed (as one can easily see) and in analyzing grammars there is no choice between doing backwards usual and strong returning derivation steps.

Now, for a PC grammar system, we show that all strong-returning derivations have an analyzing counterpart, and that, similarly, all analyzed strings can be generated in the strong-returning mode.

**Lemma 3.35** ([14]) If  $\Gamma$  is a PC grammar system, then  $L_{sr}(\Gamma) = L_{ana}(\Gamma)$ .

**Proof.** The relation  $(x_1, x_2, \dots, x_n) \xRightarrow{sr} (y_1, y_2, \dots, y_n)$  holds by a strong-returning rewriting step (point 1. of Definition 3.7) if and only if  $(y_1, y_2, \dots, y_n) \xRightarrow{ana} (x_1, x_2, \dots, x_n)$  holds by an analyzing rewriting step (point 1. of Definition 3.11).

Furthermore,  $(x_1, x_2, \dots, x_n) \xRightarrow{sr} (y_1, y_2, \dots, y_n)$  holds by point 2. of Definition 3.7 if and only if  $(y_1, y_2, \dots, y_n) \xRightarrow{ana} (x_1, x_2, \dots, x_n)$  holds by point 2. of Definition 3.11. There are no query symbols present so terminal strings are changed to the axiom, or axioms are changed to terminal strings.

Moreover,  $(x_1, x_2, \dots, x_n) \xRightarrow{sr} (y_1, y_2, \dots, y_n)$  holds by a communication step (point 3. of Definition 3.7) if and only if  $(y_1, y_2, \dots, y_n) \xRightarrow{ana} (x_1, x_2, \dots, x_n)$  holds by point 3. of Definition 3.11. To see this, consider the following. During a generating communication step all query symbols occurring in a sentential form are replaced with the requested strings if these strings do not contain further queries. If some of them do, then none of the query symbols in this certain string can be replaced. The sentential forms of those components which were able to send their strings are changed to the axiom. In an analyzing communication step all components which have sentential forms not containing query symbols are able to distribute subwords of their strings among those components which have the axiom as sentential form.  $\square$

Now as a consequence of Lemma 3.33, Lemma 3.34 and Lemma 3.35, we obtain the following theorem.

**Theorem 3.36** ([14])  $\mathcal{L}(\text{RE}) = \mathcal{L}^{\text{gen}}(\text{PC}_{\infty}\text{CF}) = \mathcal{L}^{\text{gen}}(\text{PC}_{\infty}\text{CF}, sr) = \mathcal{L}^{\text{ana}}(\text{PC}_{\infty}\text{CF})$ .

Note that with respect to analysis we have only considered non-centralized returning PC grammar systems. In a certain sense this is natural by the definition of strong return. However, we have not taken into consideration analyzing PC grammar systems in the centralized and/or non-returning case.

## 3.2 Accepting Grammars and the LBA Problem

In Section 3.1.2, it is proved that programmed grammars with  $\lambda$ -free context-free productions and with appearance checking features describe the family of context-sensitive languages if they are used as accepting devices. In other words, these acceptors recognize the same languages as linear bounded automata (LBAs) do, that is, non-deterministic Turing machines with a linear bounded work tape.

Let us consider the family of languages which can be accepted by deterministic linear-bounded automata, which we call *deterministic context-sensitive languages*. It is denoted by  $\mathcal{L}(\text{DCS})$ . Then the chain of the Chomsky hierarchy can be refined by

$$\mathcal{L}(\text{CF}) \subset \mathcal{L}(\text{DCS}) \subseteq \mathcal{L}(\text{CS}).$$

Now, the LBA problem can be expressed as the question whether the inclusion  $\mathcal{L}(\text{DCS}) \subseteq \mathcal{L}(\text{CS})$  is strict.

In order to contribute to the discussion of the LBA problem, it is natural to search for appropriate restrictions which can be attached to accepting  $\lambda$ -free programmed grammars such that they become acceptors for the class of *deterministic* context-sensitive languages, that



is, languages recognizable by deterministic LBAs. As a first step, we restrict the grammars to leftmost derivations. But for a complete elimination of non-determinism in the derivations of programmed grammars, one additionally has to restrict the number of possible choices of productions for continuing a derivation of a programmed grammar in a certain situation, that is, after a certain production has been applied leading to a certain sentential form. For this, we introduce a measure of descriptive complexity for programmed grammars, the degree of non-regulation, which reflects this number of possible choices.

This section is organized as follows. In Subsection 3.2.1 we provide the necessary definitions and notations, in particular of leftmost derivations of programmed grammars and the degree of non-regulation. We examine the cases in which the restriction to leftmost derivations changes the descriptive power of the underlying type of programmed grammars. Moreover, the hierarchies induced by the degree of non-regulation are investigated for generating and accepting programmed grammars with several types of core rules. In Subsection 3.2.2, the correspondingly defined families of languages are investigated, where the focus is on their relationship to the families of context-sensitive and deterministic context-sensitive languages. In the end, we obtain three different characterizations of the LBA problem in terms of programmed grammars, that is, the question of whether or not deterministic and non-deterministic linear bounded automata are of the same power is shown to be equivalent to the question of whether or not certain variants of programmed grammars are equally powerful. More on the LBA problem can be found in, for instance, [81, 111]—the latter reference refers to a grammatical characterization of the LBA problem in terms of Lindenmayer systems. Finally, in Subsection 3.2.3 we compare the variants of programmed grammars with regulation, that is, whose degree of non-regulation is equal to one, with respect to their descriptive power.

### 3.2.1 Restricting Nondeterminism in Programmed Grammars

#### Leftmost Derivations

Let  $G = (V_N, V_T, P, S)$  be a programmed grammar as in Definition 2.2. A derivation according to  $G$  is said to be *leftmost* if each rule is either applied in a way such that it replaces the leftmost occurrence of its left-hand side or it is applied in appearance checking mode.<sup>6</sup> By  $L_{\text{gen}}(G\text{-left})$  (or  $L_{\text{acc}}(G\text{-left})$ ) we denote the language generated (accepted, respectively) by  $G$  in this leftmost manner. In order to denote the corresponding families of languages obtained by leftmost derivations we add *left* to the first component of our notation, for example, leading to  $\mathcal{L}^{\text{gen}}(\text{P-left}, \text{CF}, \text{ac})$ .

Let us recall some results for the generating case [41] and extend them to further cases.

**Lemma 3.37 ([21])** (i) For  $X \in \{\text{CF}, \text{CF}-\lambda\}$ , we have

$$\mathcal{L}^{\text{gen}}(\text{P}, X[, \text{ac}]) \subseteq \mathcal{L}^{\text{gen}}(\text{P-left}, X[, \text{ac}]).$$

(ii) For  $X \in \{\text{REG}, \text{CS}, \text{MON}, \text{RE}\}$ ,

$$\mathcal{L}^{\text{gen}}(\text{P}, X) = \mathcal{L}^{\text{gen}}(\text{P-left}, X) = \mathcal{L}^{\text{gen}}(\text{P}, X, \text{ac}) = \mathcal{L}^{\text{gen}}(\text{P-left}, X, \text{ac}) = \mathcal{L}(X).$$

**Proof.** For (i) see [41, Lemma 1.4.8] and its proof. Statement (ii) is seen as follows: For  $X \in \{\text{REG}, \text{CS}, \text{RE}\}$ , it is proved that

$$\mathcal{L}^{\text{gen}}(\text{P}, X) = \mathcal{L}^{\text{gen}}(\text{P}, X, \text{ac}) = \mathcal{L}(X)$$

---

<sup>6</sup>Note that this definition corresponds to the definition of leftmost derivations of type 3 in [41].

in [41]. Note that  $\mathcal{L}^{\text{gen}}(\text{P}, \text{CS}) \subseteq \mathcal{L}^{\text{gen}}(\text{P}, \text{MON}) \subseteq \mathcal{L}^{\text{gen}}(\text{P}, \text{MON}, \text{ac})$  holds by definition and  $\mathcal{L}^{\text{gen}}(\text{P}, \text{MON}, \text{ac}) \subseteq \mathcal{L}(\text{CS})$  can easily be shown by LBA construction. Thus, these equalities also hold for  $X = \text{MON}$ .

Therefore, it is only left to prove that  $\mathcal{L}^{\text{gen}}(\text{P}, X[, \text{ac}]) = \mathcal{L}^{\text{gen}}(\text{P-left}, X[, \text{ac}])$ , for  $X \in \{\text{REG}, \text{CS}, \text{MON}, \text{RE}\}$ . In case of right-linear core rules this equality is obvious because there is at most one nonterminal symbol in every sentential form. In order to show  $\mathcal{L}^{\text{gen}}(\text{P}, X[, \text{ac}]) \subseteq \mathcal{L}^{\text{gen}}(\text{P-left}, X[, \text{ac}])$ , for  $X \in \{\text{CS}, \text{MON}, \text{RE}\}$ , we slightly modify the idea of the proof given for the context-free case in [41, Lemma 1.4.8]: Let us consider the programmed grammar  $G = (V_N, V_T, P, S)$  with productions of type CS, MON, or RE; with each rule of the form  $(r : \alpha \rightarrow \beta, \sigma(r), \phi(r)) \in P$ , we associate the productions

$$\begin{aligned} & (r : \alpha \rightarrow \alpha, \{r', r''\}, \phi(r)), \\ & (r' : A \rightarrow A', \{r', r''\}, \emptyset) \quad \text{if } \alpha = uA\gamma \text{ with } u \in V_T^*, A \in V_N, \\ & \quad \quad \quad \gamma \in (V_N \cup V_T)^*, \\ & (r'' : \alpha \rightarrow \beta, \{r'''\}, \emptyset), \\ & (r''' : A' \rightarrow A, \{r'''\}, \sigma(r)). \end{aligned}$$

The rule  $\alpha \rightarrow \alpha$  at label  $r$  is used to verify whether the left hand-side  $\alpha$  appears in the current sentential form. If this is not the case, then the simulation is continued at a label from  $\phi(r)$ . Otherwise, some nonterminals  $A$  are coloured to  $A'$  at label  $r'$ . This is done in order to be able to choose some appearance of  $\alpha$  in a left-most fashion. Finally, rule  $\alpha \rightarrow \beta$  at  $r''$  simulates the original rule application and  $A' \rightarrow A$  at  $r'''$  does the necessary recolouring and continues with a label from  $\sigma(r)$ .

If  $G$  is without appearance checking, the modification of the appropriate proof given in [41, Lemma 1.4.8] is analogous. For the remaining part of the construction cf. [41, Lemma 1.4.8].

Now, the proof is finished by showing the converse inclusions via LBA or Turing machine constructions, respectively.  $\square$

It is unknown whether the inclusion stated in (i) of Lemma 3.37 is strict.

Now, we turn to accepting grammars. In the case of right-linear core rules, accepting  $\lambda$ -productions can be applied only in the first step of any derivation, which yields the axiom. Therefore, by a standard argument in the theory of formal grammars, they can be substituted. Hence, (accepting) programmed grammars with right-linear core rules can be assumed to have no  $\lambda$ -productions without loss of generality. If accepting grammars in general perform leftmost derivations, then accepting  $\lambda$ -productions, that is, productions of the form  $\lambda \rightarrow u$  can be applied only at the left margin of any sentential form. Thus, even in the case of Chomsky grammars, leftmost derivations of generative grammars cannot be mimicked by the dual accepting ones, if  $\lambda$ -productions are involved. Therefore we disregard accepting grammars having  $\lambda$ -productions whenever leftmost derivations are considered. In what follows, whenever accepting programmed grammars with leftmost derivations are considered, we will restrict ourselves to the cases with core rules of types REG, CF- $\lambda$ , CS, or MON without further mentioning.

**Lemma 3.38 ([21]) (i)** We have

$$\mathcal{L}^{\text{acc}}(\text{P-left}, \text{REG}[, \text{ac}]) \subset \mathcal{L}^{\text{acc}}(\text{P}, \text{REG}[, \text{ac}]) = \mathcal{L}(\text{REG}).$$

(ii) For context-free core rules we have

$$\mathcal{L}^{\text{acc}}(\text{P}, \text{CF}-\lambda) \subseteq \mathcal{L}^{\text{acc}}(\text{P-left}, \text{CF}-\lambda).$$

(iii) For  $X \in \{\text{CS}, \text{MON}\}$ , we have

$$\mathcal{L}^{\text{acc}}(\text{P}, X) = \mathcal{L}^{\text{acc}}(\text{P-left}, X) = \mathcal{L}(\text{CS}).$$

(iv) For  $X \in \{\text{CF}-\lambda, \text{CS}, \text{MON}\}$ , we have

$$\mathcal{L}^{\text{acc}}(\text{P}, X, \text{ac}) = \mathcal{L}^{\text{acc}}(\text{P-left}, X, \text{ac}) = \mathcal{L}(\text{CS}).$$

**Proof.** The equalities  $\mathcal{L}^{\text{acc}}(\text{P}, \text{REG}[\text{ac}]) = \mathcal{L}(\text{REG})$  are shown by a standard simulation technique. The inclusions of the language families induced by leftmost derivations within  $\mathcal{L}^{\text{acc}}(\text{P}, \text{REG}[\text{ac}])$  are immediate by definition, as there is at most one nonterminal symbol in every sentential form of any successful derivation. The strictness follows by the observation that the regular language  $a^*$  cannot be accepted by any accepting programmed grammar with right-linear core rules<sup>7</sup> using leftmost derivations. The easy proof is left to the reader. This shows statement (i).

Concerning the other three statements, we first consider accepting programmed grammars with appearance checking. The inclusion  $\mathcal{L}(\text{CS}) \subseteq \mathcal{L}^{\text{acc}}(\text{P}, \text{CF}-\lambda, \text{ac})$  is proved, for example, in [16], and for  $X \in \{\text{CF}-\lambda, \text{CS}, \text{MON}\}$ ,  $\mathcal{L}^{\text{acc}}(\text{P}, \text{CF}-\lambda, \text{ac}) \subseteq \mathcal{L}^{\text{acc}}(\text{P}, X, \text{ac})$  holds by definition. The inclusion  $\mathcal{L}^{\text{acc}}(\text{P-left}, X, \text{ac}) \subseteq \mathcal{L}(\text{CS})$  with  $X \in \{\text{CF}-\lambda, \text{CS}, \text{MON}\}$  can be shown by LBA construction. Thus, it is only left to prove the inclusion  $\mathcal{L}^{\text{acc}}(\text{P}, X, \text{ac}) \subseteq \mathcal{L}^{\text{acc}}(\text{P-left}, X, \text{ac})$ , for  $X \in \{\text{CF}-\lambda, \text{CS}, \text{MON}\}$ .

This inclusion can be proved similarly to the proof for generating grammars. We just have to rename the terminal symbols first in order to avoid terminals on the right-hand sides of rules. More precisely, for a given accepting programmed grammar  $G = (V_N, V_T, P, S)$ , we construct an equivalent accepting programmed grammar with leftmost derivations  $G' = (V'_N, V'_T, P', S')$  as follows. Set  $V'_N = V_N \cup \{x' \mid x \in V_N \cup V_T\} \cup \{\bar{a} \mid a \in V_T\}$  and  $V'_T = V_T$ . Now, let  $\hat{\cdot} : (V_N \cup V_T)^* \rightarrow (V'_N)^*$  be the morphism defined by  $\hat{A} = A$  for  $A \in V_N$  and  $\hat{a} = \bar{a}$  for  $a \in V_T$ . Furthermore, let  $P'$  contain the following productions: For each  $a \in V_T$ , let

$$((0, a) : a \rightarrow \bar{a}, \{(0, b) \mid b \in V_T\}, \{r, r', r^{\text{ac}} \mid r \in \text{Lab}(P)\}),$$

and with each  $(r : \alpha \rightarrow \beta, \sigma(r), \phi(r)) \in P$ , we associate the productions

$$\begin{aligned} (r' : \hat{x} \rightarrow x', \{r, r'\}, \emptyset), & \quad \text{if } \alpha = x\gamma, x \in V_N \cup V_T, \gamma \in (V_N \cup V_T)^* \\ (r : \hat{\alpha} \rightarrow \hat{\beta}, \{r''\}, \emptyset), & \\ (r'' : x' \rightarrow \hat{x}, \{r''\}, \{p, p', p^{\text{ac}} \mid p \in \sigma(r)\}), & \\ (r^{\text{ac}} : \hat{\alpha} \rightarrow \hat{\beta}, \emptyset, \{p, p', p^{\text{ac}} \mid p \in \phi(r)\}). & \end{aligned}$$

Clearly,  $L_{\text{acc}}(G') = L_{\text{acc}}(G)$ . Note, that we have only productions of type  $X$  if every (given) rule  $\alpha \rightarrow \beta$  is of type  $X$ .

If appearance checking features are not involved, the necessary modifications parallel the ones given in [41] for the generating case: For  $a \in V_T$ , let

$$((0, a) : a \rightarrow \bar{a}, \{(0, b) \mid b \in V_T\} \cup \{r, r'\} \mid r \in \text{Lab}(P)\}, \emptyset),$$

<sup>7</sup>Note that we would have equality instead of the strict inclusion, if we considered left-linear core rules instead of right-linear ones.

and with each  $(r : \alpha \rightarrow \beta, \sigma(r), \emptyset) \in P$ , we associate the productions

$$\begin{aligned} (r' : \hat{x} \rightarrow x', \{r, r'\}, \emptyset), & \quad \text{if } \alpha = x\gamma, x \in V_N \cup V_T, \gamma \in (V_N \cup V_T)^* \\ (r : \hat{\alpha} \rightarrow \hat{\beta}, \{r''\} \cup \sigma(r) \cup \{p' \mid p \in \sigma(r)\}, \emptyset), \\ (r'' : x' \rightarrow \hat{x}, \{r''\} \cup \sigma(r) \cup \{p' \mid p \in \sigma(r)\}, \emptyset). \end{aligned}$$

Since the productions labeled by  $(0, a)$ ,  $a \in V_T$ , are the only ones replacing terminal symbols, any successful derivation must replace all terminal symbols before some productions with labels  $r$ ,  $r'$  or  $r''$  are applied. A similar argument shows that the remaining productions must be used in an appropriate order. Hence,  $L_{\text{acc}}(G') = L_{\text{acc}}(G)$ .

The proof is completed with the observation: (1)  $\mathcal{L}^{\text{acc}}(\text{P-left}, X) \subseteq \mathcal{L}(\text{CS})$  by LBA construction and (2)  $\mathcal{L}(\text{CS}) \subseteq \mathcal{L}^{\text{acc}}(\text{P}, X)$  by definition, for  $X \in \{\text{CS}, \text{MON}\}$ .  $\square$

**Example 3.39 ([21])** Consider the following accepting programmed context-free grammar  $G = (\{S, A\}, \{a\}, P, S)$ , where  $P$  contains the following productions:

$$\begin{aligned} (r_1 : a \rightarrow S, \{r_1\}, \emptyset) \\ (r_2 : a^2 \rightarrow S, \{r_2\}, \{r_3\}) \\ (r_3 : S \rightarrow A, \{r_3\}, \{r_4\}) \\ (r_4 : A^2 \rightarrow S, \{r_4\}, \{r_3\}). \end{aligned}$$

Then it is easily seen that  $L_{\text{acc}}(G)$  equals the non-context-free language  $\{a^{2^n} \mid n \geq 0\}$ . A successful sample derivation for the word  $a^4$  is

$$a^4 \xrightarrow[r_2]{} a^2 S \xrightarrow[r_2]{} SS \xrightarrow[r_3]{} AS \xrightarrow[r_3]{} AA \xrightarrow[r_4]{} S,$$

while the derivation

$$a^4 \xrightarrow[r_2]{} aSa \xrightarrow[r_3]{} aAa \xrightarrow[r_4]{} aAa \xrightarrow[r_3]{} \dots$$

runs forever, because the first step of the derivation prevents that all  $a$ 's can be rewritten accordingly. According to the above given construction we associate with  $(r_2 : a^2 \rightarrow S, \{r_2\}, \{r_3\})$  the productions

$$\begin{aligned} (r'_2 : \bar{a} \rightarrow a', \{r_2, r'_2\}, \emptyset), \\ (r_2 : \bar{a}\bar{a} \rightarrow S, \{r'_2\}, \emptyset), \\ (r''_2 : a' \rightarrow \bar{a}, \{r'_2\}, \{r_2, r'_2, r_2^{\text{ac}}\}), \\ (r_2^{\text{ac}} : \bar{a}\bar{a} \rightarrow S, \emptyset, \{r_3, r'_3, r_3^{\text{ac}}\}). \end{aligned}$$

Thus, the original derivation  $a^4 \xrightarrow[r_2]{} a^2 S$  is simulated on the word  $\bar{a}^4$  as follows

$$\bar{a}^4 \xrightarrow[r'_2]{} a'\bar{a}^3 \xrightarrow[r'_2]{} a'a'\bar{a}^2 \xrightarrow[r_2]{} a'a'S \xrightarrow[r'_2]{} \bar{a}a'S \xrightarrow[r'_2]{} \bar{a}\bar{a}S.$$

### The Degree of Non-Regulation

Programmed grammars possess a two-fold non-deterministic feature. In general, one can first select the production to be applied from the success or the failure fields. Secondly, one can

choose the occurrence of the left-hand side of the selected production in the sentential form which is to be replaced. By the restriction to leftmost derivations, the second nondeterminism is lost. In order to get rid of the nondeterminism in derivations of programmed grammars entirely, one could restrict the success and failure fields to singleton sets. In order to follow a more general approach, we consider the following measure of syntactic complexity that has been introduced in [8].

Let  $G = (V_N, V_T, P, S)$  be a programmed grammar (with or without appearance checking) with productions of type  $X$ , for  $X \in \{\text{REG}, \text{CF}, \text{CF}-\lambda, \text{CS}, \text{MON}, \text{RE}\}$ , in generating or in accepting mode. The *degree of non-regulation*  $Rg(G)$  of  $G$  is defined in the following way: For any production  $(r : \alpha \rightarrow \beta, \sigma(r), \phi(r))$  in  $P$ , we set

$$Rg(r) = \max\{|\sigma(r)|, |\phi(r)|\}$$

and

$$Rg(G) = \max\{Rg(r) \mid r \in \text{Lab}(P)\}.$$

For a language  $L \in \mathcal{L}^{\text{gen}}(\text{P}[-\text{left}], X[, \text{ac}])$ , we define the degree of non-regulation with respect to  $(\text{P}^{\text{gen}}[-\text{left}], X[, \text{ac}])$  as

$$Rg_{(\text{P}^{\text{gen}}[-\text{left}], X[, \text{ac}])}(L) = \min\{Rg(G) \mid G \text{ is a generating } (\text{P}, X[, \text{ac}])\text{-} \\ \text{grammar and } L_{\text{gen}}(G[-\text{left}]) = L\}.$$

The degree of non-regulation of a language in the family  $\mathcal{L}^{\text{acc}}(\text{P}[-\text{left}], X[, \text{ac}])$  with respect to  $(\text{P}^{\text{acc}}[-\text{left}], X[, \text{ac}])$  is defined analogously. Furthermore, we set

$$\begin{aligned} \mathcal{L}_n^{\text{gen}}(\text{P}[-\text{left}], X[, \text{ac}]) &= \{L \mid L \in \mathcal{L}^{\text{gen}}(\text{P}[-\text{left}], X[, \text{ac}]) \\ &\quad \text{and } Rg_{(\text{P}^{\text{gen}}[-\text{left}], X[, \text{ac}])}(L) \leq n\}, \\ \mathcal{L}_n^{\text{acc}}(\text{P}[-\text{left}], X[, \text{ac}]) &= \{L \mid L \in \mathcal{L}^{\text{acc}}(\text{P}[-\text{left}], X[, \text{ac}]) \\ &\quad \text{and } Rg_{(\text{P}^{\text{acc}}[-\text{left}], X[, \text{ac}])}(L) \leq n\}. \end{aligned}$$

The inclusions

$$\mathcal{L}_n^{\text{gen}}(\text{P}[-\text{left}], X[, \text{ac}]) \subseteq \mathcal{L}_{n+1}^{\text{gen}}(\text{P}[-\text{left}], X[, \text{ac}]) \subseteq \mathcal{L}^{\text{gen}}(\text{P}[-\text{left}], X[, \text{ac}])$$

and

$$\mathcal{L}_n^{\text{acc}}(\text{P}[-\text{left}], X[, \text{ac}]) \subseteq \mathcal{L}_{n+1}^{\text{acc}}(\text{P}[-\text{left}], X[, \text{ac}]) \subseteq \mathcal{L}^{\text{acc}}(\text{P}[-\text{left}], X[, \text{ac}])$$

trivially hold for  $n \geq 1$ . Obviously, the degree of non-regulation of a programmed grammar is a measure for the maximum number of possibilities to continue a derivation of the form  $z \xrightarrow[p]{*} z' \Longrightarrow w$  according to  $G$ , for some sentential forms  $z, z', w$  and a label  $p \in \text{Lab}(P)$ .

**Lemma 3.40 ([21])** Let  $X \in \{\text{REG}, \text{CF}, \text{CF}-\lambda, \text{CS}, \text{MON}, \text{RE}\}$ . Then

(i)  $\mathcal{L}_2^{\text{gen}}(\text{P}, X[, \text{ac}]) = \mathcal{L}^{\text{gen}}(\text{P}, X[, \text{ac}])$  and

(ii)  $\mathcal{L}_2^{\text{acc}}(\text{P}, X[, \text{ac}]) = \mathcal{L}^{\text{acc}}(\text{P}, X[, \text{ac}])$ .

**Proof.** First, we consider a generating programmed grammar  $G = (V_N, V_T, P, S)$ . With each production

$$(r : \alpha \rightarrow \beta, \{s_1, s_2, \dots, s_\ell\}, \{t_1, t_2, \dots, t_k\}) \in P, \quad \ell > 2 \text{ or } k > 2,$$

we associate the following groups of productions:

$$\begin{aligned} (r : \alpha \rightarrow \alpha, \{r'_1, r_2\}, \{t_1, r_2\}), \\ (r'_1 : \alpha \rightarrow \beta, \{s_1\}, \emptyset), \end{aligned}$$

if  $\ell \neq k$ , for  $1 < i \leq \min\{\ell, k\}$ ,

$$\begin{aligned} (r_i : \alpha \rightarrow \alpha, \{r'_i, r_{i+1}\}, \{t_i, r_{i+1}\}), \\ (r'_i : \alpha \rightarrow \beta, \{s_i\}, \emptyset), \end{aligned}$$

and, if  $\ell < k$ , for  $\ell < j < k$ ,

$$\begin{aligned} (r_j : \alpha \rightarrow \alpha, \emptyset, \{t_j, r_{j+1}\}), \quad \text{and} \\ (r_k : \alpha \rightarrow \alpha, \emptyset, \{t_k\}), \end{aligned}$$

if  $k < \ell$ , for  $k < j < \ell$ ,

$$\begin{aligned} (r_j : \alpha \rightarrow \alpha, \{r'_j, r_{j+1}\}, \emptyset), \quad \text{and} \\ (r'_j : \alpha \rightarrow \beta, \{s_j\}, \emptyset) \\ (r_\ell : \alpha \rightarrow \beta, \{s_\ell\}, \emptyset). \end{aligned}$$

Finally, in the case of  $\ell = k$ , we perform the same construction but we set

$$(r_k : \alpha \rightarrow \beta, \{s_k\}, \{t_k\}).$$

If  $\sigma(r) = \emptyset$  (or  $\phi(r) = \emptyset$ ) we let the success fields (the failure fields, respectively) be empty in all these productions. Now, we construct the programmed grammar  $G' = (V_N, V_T, P', S)$  by replacing each rule  $r \in P$  with  $Rg(r) > 2$  by the corresponding group of productions as listed above in order to obtain  $P'$ . Then  $G'$  is of the same type as  $G$  and  $L_{\text{gen}}(G') = L_{\text{gen}}(G)$  holds.

In the accepting case, let  $G' = (V'_N, V_T, P', S)$ , where  $V'_N = V_N \cup \{x' \mid x \in V_N \cup V_T\}$ ,  $V_N \cap \{x' \mid x \in V_N \cup V_T\} = \emptyset$ , and  $P'$  is constructed as above under the following modifications: instead of  $(r : \alpha \rightarrow \alpha, \{r'_1, r_2\}, \{t_1, r_2\})$  we take  $(r : \alpha \rightarrow \beta', \{r'_1, r_2\}, \{t_1, r_2\})$ , where

$$\beta' = \begin{cases} A' & \text{if } G \text{ is right-linear or context-free and } \beta = A \\ \gamma_1 A' \gamma_2 & \text{if } G \text{ is context-sensitive and } \alpha = \gamma_1 v \gamma_2, \beta = \gamma_1 A \gamma_2 \\ x'_1 \dots x'_m & \text{if } G \text{ is monotone or of type-0 and } \beta = x_1 \dots x_m. \end{cases}$$

Moreover, we replace any further occurrence of  $\alpha \rightarrow \alpha$  by  $\beta' \rightarrow \beta'$  and any occurrence of  $\alpha \rightarrow \beta$  by  $\beta' \rightarrow \beta$  in the above groups of productions associated to rule  $r$ . Thus, we obtain only accepting rules of the same type and we have  $L_{\text{acc}}(G') = L_{\text{acc}}(G)$ .  $\square$

Let us mention that a similar construction is possible for the special case of programmed grammars with unconditional transfer (for a definition cf. [41]). Moreover, the constructions given in the proof of Lemma 3.40 also work for the case of leftmost derivations, that is,  $L_{\text{gen}}(G'\text{-left}) = L_{\text{gen}}(G\text{-left})$  ( $L_{\text{acc}}(G'\text{-left}) = L_{\text{acc}}(G\text{-left})$ , respectively).

**Corollary 3.41 ([21]) (i)** For  $X \in \{\text{REG}, \text{CF}, \text{CF}-\lambda, \text{CS}, \text{MON}, \text{RE}\}$ , we have

$$\mathcal{L}_2^{\text{gen}}(\text{P-left}, X[, \text{ac}]) = \mathcal{L}^{\text{gen}}(\text{P-left}, X[, \text{ac}]),$$

(ii) For  $X \in \{\text{REG}, \text{CF}-\lambda, \text{CS}, \text{MON}\}$ , we have

$$\mathcal{L}_2^{\text{acc}}(\text{P-left}, X[, \text{ac}]) = \mathcal{L}^{\text{acc}}(\text{P-left}, X[, \text{ac}]).$$

For the cases  $X \in \{\text{CS}, \text{MON}, \text{RE}\}$  *without* left-most derivations but with appearance checking, we can even improve the result.

**Lemma 3.42 ([21])** Let  $X \in \{\text{CS}, \text{MON}, \text{RE}\}$ . Then

(i)  $\mathcal{L}_1^{\text{gen}}(\text{P}, X, \text{ac}) = \mathcal{L}(X)$  and

(ii)  $\mathcal{L}_1^{\text{acc}}(\text{P}, X, \text{ac}) = \mathcal{L}(X)$ .

**Proof.** We only proof the first statement. The accepting case can be proven with a similar construction.

Because of  $\mathcal{L}(X) = \mathcal{L}^{\text{gen}}(\text{P}, X, \text{ac})$ , we only have to show the inclusion from right to left. We argue as follows: By standard arguments the family  $\mathcal{L}_1^{\text{gen}}(\text{P}, X, \text{ac})$  is closed under union and includes the finite languages. Let  $L \subseteq V_T^*$  be in  $\mathcal{L}(X)$ , then

$$L = \bigcup_{a,b,c \in V_T} (\delta_{abc}(L) \cdot abc) \cup (L \cap V_T^3) \cup (L \cap V_T^2) \cup (L \cap V_T) \cup (L \cap \{\lambda\}),$$

where  $\delta_{abc}(L) = \{w \in V_T^+ \mid wabc \in L\}$ . Since  $L$  is in  $\mathcal{L}(X)$ , the language  $\delta_{abc}(L)$  is in  $\mathcal{L}(X)$  due to the closure of that family under derivatives. Thus, for the proof of the present assertion, it is sufficient to show that  $\delta_{abc}(L) \cdot abc$  is in  $\mathcal{L}_1^{\text{gen}}(\text{P}, X, \text{ac})$ , provided that  $\delta_{abc}(L)$  is in  $\mathcal{L}(X)$ .

Let  $G_{abc} = (V_N, V_T, P, S)$  be a grammar of type  $X$  generating  $\delta_{abc}(L)$ . Assume that  $P = \{\alpha_1 \rightarrow \beta_1, \alpha_2 \rightarrow \beta_2, \dots, \alpha_n \rightarrow \beta_n\}$ . Let  $S'$ ,  $X$ , and  $X'$  be new symbols not contained in  $V_N \cup V_T$ . We construct a programmed grammar  $G = (V_N \cup \{S', X, X'\}, V_T, P', S')$  with labels

$$\{[init]\} \cup \{[t, 1], [t, 2], [t, 3]\} \cup \{[r_i, j] \mid 1 \leq i \leq n \text{ and } 1 \leq j \leq 7\}$$

as follows: To start the derivation we are going to use the production

$$([init] : S' \rightarrow SXXX, \{[r_1, 1]\}, \emptyset).$$

Then, for every  $1 \leq i \leq n$ , let the following rules be given.

$$\begin{aligned} ([r_i, 1] & : XX \rightarrow XX', \{[r_i, 2]\}, \emptyset) \\ ([r_i, 2] & : XXX' \rightarrow XXX', \{[r_i, 3]\}, \{[r_i, 4]\}) \\ ([r_i, 3] & : \alpha_i \rightarrow \beta_i, \{[r_i, 4]\}, \emptyset) \\ ([r_i, 4] & : XX' \rightarrow XX, \{[r_i, 5]\}, \emptyset) \\ ([r_i, 5] & : XX \rightarrow XX', \{[r_i, 6]\}, \emptyset) \\ ([r_i, 6] & : XXX' \rightarrow XXX', \{[r_i, 7]\}, \{[t, 1]\}) \\ ([r_i, 7] & : XX' \rightarrow XX, \{[r_j, 1] \mid j = 1 \text{ if } i = n, \\ & \text{and } j = i + 1 \text{ otherwise}\}, \emptyset) \end{aligned}$$

The  $XXX$  symbols in the sentential form are used as a switch in order to (1) apply the actual rule  $\alpha_i \rightarrow \beta_i$  to the sentential form—rules with labels  $[r_i, 1]$ ,  $[r_i, 2]$ , and  $[r_i, 3]$ —and (2) to start the termination of the derivation process—rules with labels  $[r_i, 5]$  and  $[r_i, 6]$ . To be more precise, one can do the following derivations on a sentential form  $\alpha XXX$ :

- (i) Apply the rule  $\alpha_i \rightarrow \beta_i$  and continue with the next label  $[r_1, 1]$ , if  $i = n$ , and  $[r_{i+1}, 1]$  otherwise:

$$\begin{aligned} \alpha XXX &\xRightarrow{[r_i, 1]} \alpha XXX' \\ &\xRightarrow{[r_i, 2]} \alpha XXX' \xRightarrow{[r_i, 3]} \alpha' XXX' \xRightarrow{[r_i, 4]} \alpha' XXX \\ &\xRightarrow{[r_i, 5]} \alpha' XXX' \xRightarrow{[r_i, 6]} \alpha' XXX' \xRightarrow{[r_i, 7]} \alpha' XXX, \end{aligned}$$

where  $\alpha'$  is derived from  $\alpha$  by the rule under consideration.

- (ii) Apply the rule  $\alpha_i \rightarrow \beta_i$  and start the termination process by continuing at  $[t, 1]$ :

$$\begin{aligned} \alpha XXX &\xRightarrow{[r_i, 1]} \alpha XXX' \xRightarrow{[r_i, 2]} \alpha XXX' \xRightarrow{[r_i, 3]} \alpha' XXX' \\ &\xRightarrow{[r_i, 4]} \alpha' XXX' \xRightarrow{[r_i, 5]} \alpha' XXX' \xRightarrow{[r_i, 6]} \alpha' XXX' \xRightarrow{[r_i, 6]} \alpha' XXX', \end{aligned}$$

where  $\alpha'$  is as described above.

- (iii) Do not apply the rule  $\alpha_i \rightarrow \beta_i$  and continue with the next label  $[r_1, 1]$ , if  $i = n$ , and  $[r_{i+1}, 1]$  otherwise:

$$\begin{aligned} \alpha XXX &\xRightarrow{[r_i, 1]} \alpha XXX' \xRightarrow{[r_i, 2]} \alpha XXX' \xRightarrow{[r_i, 4]} \alpha XXX \\ &\xRightarrow{[r_i, 5]} \alpha XXX' \xRightarrow{[r_i, 6]} \alpha XXX' \xRightarrow{[r_i, 7]} \alpha XXX. \end{aligned}$$

- (iv) Do not apply the rule  $\alpha_i \rightarrow \beta_i$  and start the termination process by continuing with label  $[t, 1]$ :

$$\begin{aligned} \alpha XXX &\xRightarrow{[r_i, 1]} \alpha XXX' \xRightarrow{[r_i, 2]} \alpha XXX' \\ &\xRightarrow{[r_i, 4]} \alpha XXX \xRightarrow{[r_i, 5]} \alpha XXX' \xRightarrow{[r_i, 6]} \alpha XXX'. \end{aligned}$$

Since the rules are checked in sequence, we can simulate each derivation of the original grammar.

Finally, in order to terminate the derivation process we use

$$\begin{aligned} ([t, 1] &: XX'X \rightarrow XX'c, \{[t, 2]\}, \emptyset), \\ ([t, 2] &: XX'c \rightarrow Xbc, \{[t, 3]\}, \emptyset), \\ ([t, 3] &: Xbc \rightarrow abc, \{[t, 3]\}, \emptyset). \end{aligned}$$

This completes the construction of  $G$ . Obviously,  $Rg(G) = 1$ , and it is easy to see that  $L(G) = \delta_{abc}(L) \cdot abc$ .  $\square$

What about the degree 1 of non-regulation for programmed grammars with rules of the types  $X$  with  $X \in \{\text{REG}, \text{CF}, \text{CF}-\lambda\}$ ? The cases of generating programmed grammars (with or without appearance checking) and accepting programmed grammars without appearance checking are treated in Section 3.2.3. Concerning accepting programmed grammars with appearance checking, the context-free case is investigated in the next subsection, where a close relation to the LBA problem is shown. In the remainder of this subsection we consider the case of right-linear core rules in accepting programmed grammars with appearance checking.



**Lemma 3.43** ([21])  $\mathcal{L}_1^{\text{acc}}(\text{P}, \text{REG}, \text{ac}) = \mathcal{L}(\text{REG})$ .

**Proof.** The inclusion from left to right follows by Lemma 3.38. The converse inclusion is seen as follows: As the family of regular languages is closed with respect to mirror image, and  $(L^R)^R = L$  for any language  $L$ , we have  $\{L \mid L^R \in \mathcal{L}(\text{REG})\} = \mathcal{L}(\text{REG})$ . Therefore, it is sufficient to simulate deterministic finite automata reading the input one-way from right to left, which will be called RL-DFA's in the following.

Let  $M = (Q, \Sigma, \delta, q_0, F)$  be an RL-DFA, where  $Q$  is the set of states,  $\Sigma$  is the alphabet of input symbols,  $q_0 \in Q$  is the initial state,  $F \subseteq Q$  is the set of accepting states, and  $\delta : Q \times \Sigma \rightarrow Q$  is the transition function. Let  $\Sigma = \{a_1, a_2, \dots, a_n\}$ ,  $Q = \{q_0, q_1, \dots, q_m\}$ , and assume that  $q_0$  is the unique initial state. We construct a programmed grammar  $G = (Q \cup \{S\}, \Sigma, P, S)$ ,  $S \notin Q$ , with labels

$$\{[init]\} \cup \{[i, j] \mid 0 \leq i \leq m \text{ and } 1 \leq j \leq n\} \cup \{[term_i] \mid 0 \leq i \leq m\}$$

and where  $P$  contains the following groups of productions.

(1) The derivation is started with the rule

$$([init] : \lambda \rightarrow q_0, \{[0, 1]\}, \emptyset).$$

(2) The simulation of the finite automaton  $A$  is done by the following rules. For every  $0 \leq i \leq m$  define

$$([i, j] : a_j q_i \rightarrow q_k, \{[k, 1]\}, \{[i, j + 1]\}),$$

if  $1 \leq j < n$  and  $\delta(q_i, a_j) = q_k$ . Moreover let

$$([i, n] : a_n q_i \rightarrow q_k, \{[k, 1]\}, \{[term_i]\}),$$

where  $\delta(q_i, a_n) = q_k$ .

(3) In order to terminate the derivation one of the following rules can be used:

$$\begin{aligned} ([term_i] : q_i \rightarrow S, \{[term_i]\}, \emptyset), & \quad 0 \leq i \leq m \text{ and } q_i \in F, \\ ([term_i] : q_i \rightarrow q_i, \{[term_i]\}, \emptyset), & \quad 0 \leq i \leq m \text{ and } q_i \notin F. \end{aligned}$$

This completes the description of the accepting programmed right-linear grammar  $G$  with  $Rg(G) = 1$ . It is easy to see that  $L_{\text{acc}}(G)$  equals the language accepted by the RL-DFA  $M$ .  $\square$

### 3.2.2 On Context-Sensitive and Deterministic Context-Sensitive Languages

First, we present some characterizations of the family of context-sensitive languages with the help of programmed grammars. From the literature, the following characterizations are known. In [41] it has been shown that, for  $X \in \{\text{CS}, \text{MON}\}$ ,

$$\mathcal{L}^{\text{gen}}(\text{P}, X) = \mathcal{L}^{\text{gen}}(\text{P}, X, \text{ac}) = \mathcal{L}(\text{CS})$$

holds. Furthermore, in Section 3.1.2 it is proved that the family of *accepting* programmed grammars with appearance checking and  $\lambda$ -free context-free core rules describes exactly the family  $\mathcal{L}(\text{CS})$ . If we take into consideration the results of the previous subsection, we can give some further characterizations of this language family.

**Corollary 3.44** ([21]) For  $X \in \{\text{CF-}\lambda, \text{CS}, \text{MON}\}$  and  $Y \in \{\text{CS}, \text{MON}\}$ , all the following families of languages are equal to  $\mathcal{L}(\text{CS})$ :

$$(i) \quad \mathcal{L}^{\text{acc}}(\text{P}, X, \text{ac}) = \mathcal{L}^{\text{acc}}(\text{P-left}, X, \text{ac}) = \mathcal{L}_2^{\text{acc}}(\text{P}, X, \text{ac}) = \mathcal{L}_2^{\text{acc}}(\text{P-left}, X, \text{ac}).$$

$$(ii) \quad \mathcal{L}^{\text{acc}}(\text{P}, Y) = \mathcal{L}^{\text{acc}}(\text{P-left}, Y) = \mathcal{L}_2^{\text{acc}}(\text{P}, Y) = \mathcal{L}_2^{\text{acc}}(\text{P-left}, Y).$$

$$(iii) \quad \mathcal{L}_1^{\text{gen}}(\text{P}, Y, \text{ac}) = \mathcal{L}_1^{\text{acc}}(\text{P}, Y, \text{ac}).$$

**Proof.** Let  $X$  and  $Y$  be as in the statement of the corollary. From Lemma 3.38, it is known that the equalities

$$\mathcal{L}^{\text{acc}}(\text{P}, X, \text{ac}) = \mathcal{L}^{\text{acc}}(\text{P-left}, X, \text{ac}) = \mathcal{L}(\text{CS})$$

and

$$\mathcal{L}^{\text{acc}}(\text{P}, Y) = \mathcal{L}^{\text{acc}}(\text{P-left}, Y) = \mathcal{L}(\text{CS})$$

hold. Thus, together with Lemma 3.40 and Corollary 3.41, the equalities claimed in (i) and (ii) are proved. The equalities of  $\mathcal{L}_1^{\text{gen}}(\text{P}, Y, \text{ac})$  and  $\mathcal{L}_1^{\text{acc}}(\text{P}, Y, \text{ac})$  to  $\mathcal{L}(\text{CS})$  have been shown in Lemma 3.42.  $\square$

Since the restriction of the degree of non-regulation of programmed grammars to 1 means the elimination of some non-deterministic aspect, it is natural to investigate the interrelations between the families  $\mathcal{L}(\text{DCS})$  and, for example,  $\mathcal{L}_1^{\text{acc}}(\text{P[-left]}, \text{CF-}\lambda, \text{ac})$ . Here we find the following situation.

**Lemma 3.45** ([21]) For  $X \in \{\text{CF-}\lambda, \text{CS}, \text{MON}\}$ , we have

$$\mathcal{L}_1^{\text{acc}}(\text{P-left}, X, \text{ac}) \subseteq \mathcal{L}(\text{DCS}).$$

**Proof.** Let  $G = (V_N, V_T, P, S)$  be an accepting programmed grammar with productions of type  $X$  (working in leftmost mode) and  $Rg(G) = 1$ . Furthermore, let  $k = |P_T|$ , where  $P_T = \{\alpha \rightarrow \beta \in P \mid \alpha \in V_T^+\}$ . Let  $\{p_{i_1}, p_{i_2}, \dots, p_{i_k}\}$  be the set of labels of the rules in  $P_T$ . Construct a deterministic linear-bounded automaton  $M = (Q, V_T, V_N \cup V_T \cup \{\#, \$, B\}, \delta, q_0, B, F)$ , where the states are tuples with the first  $k$  components memorizing labels  $r_{i_1}, r_{i_2}, \dots, r_{i_k}$  from  $\text{Lab}(P)$ , in  $q_0$  initialized by  $r_{i_j} = p_{i_j}$ ,  $1 \leq j \leq k$ , such that  $M$  performs the following steps.

**Step 1.** Copy the input word  $w$ ,  $w \in V_T^*$ ,  $(k-1)$ -fold such that we obtain the string

$$\underbrace{\#w\#w\#w \cdots w\#w\#}_{k\text{-times } w}.$$

**Step 2.** Perform stepwise simulations of leftmost derivation steps according to  $G$  in the following way. For  $j = 1$  to  $k$ :

**2.1** Search for the leftmost occurrence of the left-hand side  $\alpha_{i_j}$  of rule  $r_{i_j}$  in the  $j$ -th subword between markers  $\#$  on the tape from the left to the right (in the  $j$ -th occurrence of  $w$ , in the beginning).

**2.2** If such substring is found and  $|\sigma(r_{i_j})| = 1$ , replace it by the right-hand side  $\beta_{i_j}$  of rule  $r_{i_j}$ , if  $|\beta_{i_j}| = |\alpha_{i_j}|$ , and replace it by  $\beta_{i_j}\$^m$  if  $|\alpha_{i_j}| - |\beta_{i_j}| = m$ ,  $m > 0$ ; change the  $j$ -th component of the current state to  $\sigma(r_{i_j})$ . If  $\#$  is reached and no subword  $\alpha_{i_j}$  has been found in the current subword change the  $j$ -th component of the state to  $\phi(r_{i_j})$  if possible and do not change the  $j$ -th component of the state otherwise.

**2.3** If  $j < k$ , move the read-write head over the first cell of the  $(j + 1)$ -st subword and proceed with 2.1 for the next  $j$ . If  $j = k$  then remove all occurrences of  $\$$  and shift the remaining parts such that we obtain a “compact” word over  $V_N \cup V_T \cup \{\#\}$ . Then move the read-write head over the first cell of the first subword.

**Step 3.** Check whether a subword  $\#S\#$  occurs in the current contents of the tape. If yes, then accept, otherwise repeat step 2 according to the currently memorized rules in the state.

Clearly, indeed  $M$  is deterministic and linear-bounded and accepts exactly the words in  $L_{\text{acc}}(G\text{-left})$ .  $\square$

Regarding the converse inclusion, we can only prove the following statement.

**Lemma 3.46** ([21]) For  $X \in \{\text{CS}, \text{MON}\}$ , we have

$$\mathcal{L}(\text{DCS}) \subseteq \mathcal{L}_1^{\text{acc}}(\text{P-left}, X, \text{ac}).$$

**Proof.** Given a deterministic linear-bounded automaton  $M$ . Without loss of generality, let  $M$  possess the following standard properties of (deterministic) LBAs. We assume that, in any computation,  $M$  first marks the leftmost symbol and the rightmost symbol of the input word with, say, superscript  $L$  and  $R$ , respectively. Then, by the theorem about tape compression (for instance, see [63, Theorem 12.1]), the number of cells used during the computation on the input word  $w$  can be assumed to be equal to  $|w|$ , that is, the read-write head is allowed to visit only the cells which are occupied by the input word at the beginning of the computation. Therefore, if a transition has to be applied where the read-write head moves to the right, then the cell to the right is occupied by a non-blank symbol, and if the read-write head has to be moved to the left, then a non-blank symbol will be met there. Moreover, we can assume that there are no transitions for final states, that is,  $M$  has no next moves whenever the input is accepted. Let  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ , where  $\Sigma = \{a_1, a_2, \dots, a_n\}$  and  $\Gamma = \{a_1, a_2, \dots, a_n, a_{n+1}, a_{n+2}, \dots, a_r\}$ . We set

$$V_\Gamma = \Gamma \cup \{x^L \mid x \in \Gamma\} \cup \{x^R \mid x \in \Gamma\}$$

and construct an accepting programmed grammar  $G = (V_N, \Sigma, P, S)$  with

$$V_N = V_\Gamma \cup (Q \times V_\Gamma) \cup \{x' \mid x \in \Sigma\} \cup \{S\}$$

(the unions being disjoint) and where  $P$  contains the following groups of productions.

- (1) At first, we allow the grammar to accept the “correct” words of length one.

$$([i] : a_i \rightarrow S, \emptyset, \emptyset), \text{ if } a_i \in \Sigma \cap L(M)$$

- (2) The grammar can guess the leftmost symbol of its input string and mark it with superscript  $L$  and can then find the rightmost symbol in order to mark it with superscript  $R$ .

$$\begin{aligned} ([init_L, i] : a_i \rightarrow a_i^L, [init, i, 1], \emptyset), & \quad 1 \leq i \leq n, \\ ([init, i, j] : a_i^L a_j \rightarrow a_i^L a'_j, [i, j, 1], [init, i, j + 1]), & \quad 1 \leq i \leq n, \\ & \quad 1 \leq j < n, \\ ([init, i, n] : a_i^L a_n \rightarrow a_i^L a'_n, [i, n, 1], \emptyset), & \quad 1 \leq i \leq n, \\ ([i, j, k] : a'_j a_k \rightarrow a_j a'_k, [i, k, 1], [i, j, k + 1]), & \quad 1 \leq i, j \leq n, \\ & \quad 1 \leq k < n, \\ ([i, j, n] : a'_j a_n \rightarrow a_j a'_n, [i, n, 1], [init_R, i, j]), & \quad 1 \leq i, j \leq n, \\ ([init_R, i, j] : a'_j \rightarrow a_j^R, [0, i], \emptyset), & \quad 1 \leq i, j \leq n \end{aligned}$$

- (3) The simulation of the work of  $M$  can be initiated by

$$([0, i] : a_i^L \rightarrow (q_0, a_i^L), [q_0, a_i^L, a_1], \emptyset), \quad 1 \leq i \leq n.$$

- (4) Any transition of  $M$  of the form  $\delta(q, \hat{a}_i) = (q', \hat{a}_k, R)$  can be simulated, where  $\hat{x} \in \{x, x^L\}$ , for  $x \in \Gamma$ .

$$\begin{aligned} ([q, \hat{a}_i, a_j] : (q, \hat{a}_i) a_j \rightarrow \hat{a}_k(q', a_j), [q', a_j, a_1], [q, \hat{a}_i, a_{j+1}]), & \quad 1 \leq j < r, \\ ([q, \hat{a}_i, a_r] : (q, \hat{a}_i) a_r \rightarrow \hat{a}_k(q', a_r), [q', a_r, a_1], [q, \hat{a}_i, a_1^R]), & \\ ([q, \hat{a}_i, a_j^R] : (q, \hat{a}_i) a_j^R \rightarrow \hat{a}_k(q', a_j^R), [q', a_j^R, a_1], [q, \hat{a}_i, a_{j+1}^R]), & \quad 1 \leq j < r, \\ ([q, \hat{a}_i, a_r^R] : (q, \hat{a}_i) a_r^R \rightarrow \hat{a}_k(q', a_r^R), [q', a_r^R, a_1], \emptyset) & \end{aligned}$$

- (5) Any transition of  $M$  of the form  $\delta(q, \bar{a}_i) = (q', \bar{a}_k, L)$  can be simulated, where  $\bar{x} \in \{x, x^R\}$ , for  $x \in \Gamma$ .

$$\begin{aligned} ([q, \bar{a}_i, a_j] : a_j(q, \bar{a}_i) \rightarrow (q', a_j) \bar{a}_k, [q', a_j, a_1], [q, \bar{a}_i, a_{j+1}]), & \quad 1 \leq j < r, \\ ([q, \bar{a}_i, a_r] : a_r(q, \bar{a}_i) \rightarrow (q', a_r) \bar{a}_k, [q', a_r, a_1], [q, \bar{a}_i, a_1^L]), & \\ ([q, \bar{a}_i, a_j^L] : a_j^L(q, \bar{a}_i) \rightarrow (q', a_j^L) \bar{a}_k, [q', a_j^L, a_1], [q, \bar{a}_i, a_{j+1}^L]), & \quad 1 \leq j < r, \\ ([q, \bar{a}_i, a_r^L] : a_r^L(q, \bar{a}_i) \rightarrow (q', a_r^L) \bar{a}_k, [q', a_r^L, a_1], \emptyset) & \end{aligned}$$

(6) In order to terminate, add the productions

$$\begin{aligned}
& ([q_f, x, a_1] : (q_f, x) \rightarrow \tilde{x}, 1, \emptyset), & \text{for all } x \in V_\Gamma, q_f \in F, \\
& & \text{where } \tilde{x} = x' \text{ if } x \in \Sigma \\
& & \text{and } \tilde{x} = x \text{ otherwise,} \\
& (i : x_i^L \rightarrow x_i^L, \langle i, 1 \rangle, i+1), & 1 \leq i < r-1, \\
& (r-1 : x_{r-1}^L \rightarrow x_{r-1}^L, \langle r-1, 1 \rangle, \langle r, 1 \rangle), \\
& (\langle i, j \rangle : x_i^L x_j \rightarrow x_j^L, \langle j, 1 \rangle, \langle i, j+1 \rangle), & 1 \leq i \leq r, 1 \leq j < r, \\
& (\langle i, r \rangle : x_i^L x_r \rightarrow x_r^L, \langle r, 1 \rangle, \langle i, 1' \rangle), & 1 \leq i \leq r, \\
& (\langle i, j' \rangle : x_i^L x'_j \rightarrow x_j^L, \langle j, 1 \rangle, \langle i, j+1' \rangle), & 1 \leq i \leq r, 1 \leq j < n, \\
& (\langle i, n' \rangle : x_i^L x'_n \rightarrow x_n^L, \langle n, 1 \rangle, \langle i, 1, R \rangle), & 1 \leq i \leq r, \\
& (\langle i, j, R \rangle : x_i^L x_j^R \rightarrow S, \emptyset, \langle i, j+1, R \rangle), & 1 \leq i \leq r, 1 \leq j < r, \\
& (\langle i, r, R \rangle : x_i^L x_r^R \rightarrow S, \emptyset, \emptyset), & 1 \leq i \leq r.
\end{aligned}$$

Note that the primed versions of the input symbols are needed here in order to have a nonterminal symbol on the right-hand sides of the productions labeled by  $[q_f, x, a_1]$ .

Given a word  $w \in \Sigma^*$ , with  $|w| \geq 2$ , the only possibility to initiate a successful derivation is to start by a rule with label  $[init_L, i]$ . If the “correct”  $i$  has been guessed, then the leftmost symbol is marked by  $L$ ; next the rightmost symbol will be found and marked by  $R$ . After application of production  $[0, i]$  the leftmost symbol carries the initial state as an additional component. Now, the transitions of  $M$  can be simulated, where the marked symbols remain on the extreme positions of the sentential form, until a sentential form is arrived which contains a symbol from  $F \times V_\Gamma$ . Then the productions of the final group allow to derive  $S$ , but only if indeed the leftmost symbol has been marked by  $L$  in the first step and  $w$  can be recognized by  $M$ . If a symbol different from the leftmost symbol of  $w$  has been marked in the first step then it is impossible to derive the axiom since letters to the left of the marked one cannot be replaced. In conclusion,  $G$  accepts (by leftmost derivations) exactly the words recognized by  $M$ . Note that  $Rg(G) = 1$  and that  $G$  has monotone core rules. By a standard trick in formal language theory, one can transform all the productions of  $G$  to context-sensitive ones without changing the degree of non-regulation. This proves the statement of the Lemma.  $\square$

The Lemmas 3.45 and 3.46 lead to the equality  $\mathcal{L}_1^{\text{acc}}(\text{P-left}, X, \text{ac}) = \mathcal{L}(\text{DCS})$ , for  $X \in \{\text{CS}, \text{MON}\}$ . Together with Corollary 3.44, we see that the question of whether or not the degree of non-regulation of accepting programmed grammars with context-sensitive or monotone productions and with leftmost derivations can be reduced to 1 without loss of descriptive capacity is equivalent to the LBA problem. In case of  $\lambda$ -free context-free core rules, the relationship to the LBA problem is a little bit weaker, since it is unknown whether the inclusion  $\mathcal{L}_1^{\text{acc}}(\text{P-left}, \text{CF}-\lambda, \text{ac}) \subseteq \mathcal{L}(\text{DCS})$  is strict.

**Corollary 3.47 ([21])** (i) For  $X \in \{\text{CS}, \text{MON}\}$ , we have

$$\mathcal{L}_1^{\text{acc}}(\text{P-left}, X, \text{ac}) = \mathcal{L}(\text{DCS}) \subseteq \mathcal{L}(\text{CS}) = \mathcal{L}_2^{\text{acc}}(\text{P-left}, X, \text{ac}).$$

(ii) Moreover,

$$\mathcal{L}_1^{\text{acc}}(\text{P-left}, \text{CF}-\lambda, \text{ac}) \subseteq \mathcal{L}(\text{DCS}) \subseteq \mathcal{L}(\text{CS}) = \mathcal{L}_2^{\text{acc}}(\text{P-left}, \text{CF}-\lambda, \text{ac}).$$

For  $X \in \{\text{CS}, \text{MON}\}$ , we find an analogous inclusion chain also without appearance checking features, both in generating and accepting mode. It is proved in the next subsection that the first inclusion is strict in both modes.

**Corollary 3.48** ([21]) For  $X \in \{\text{CS}, \text{MON}\}$ ,

$$(i) \mathcal{L}_1^{\text{gen}}(\text{P-left}, X) \subseteq \mathcal{L}(\text{DCS}) \subseteq \mathcal{L}(\text{CS}) = \mathcal{L}_2^{\text{gen}}(\text{P[-left]}, X),$$

$$(ii) \mathcal{L}_1^{\text{acc}}(\text{P-left}, X) \subseteq \mathcal{L}(\text{DCS}) \subseteq \mathcal{L}(\text{CS}) = \mathcal{L}_2^{\text{acc}}(\text{P[-left]}, X).$$

**Proof.** The idea of the proof of Lemma 3.45 can easily be transferred to languages in  $\mathcal{L}_1^{\text{gen}}(\text{P-left}, X)$ . Concerning the equalities to  $\mathcal{L}(\text{CS})$ , see the Lemmas 3.37, 3.38, and 3.40 as well as Corollary 3.41.  $\square$

Due to Lemma 3.42, we obtain another, precise characterization of the LBA problem in terms of programmed grammars.

**Corollary 3.49** ([21]) For  $X \in \{\text{CS}, \text{MON}\}$ , we have

$$\mathcal{L}_1^{\text{acc}}(\text{P-left}, X, \text{ac}) = \mathcal{L}(\text{DCS}) \subseteq \mathcal{L}(\text{CS}) = \mathcal{L}_1^{\text{acc}}(\text{P}, X, \text{ac}).$$

### 3.2.3 Programmed Grammars with Regulation

It will be shown there are variants of programmed grammars which characterize exactly the family  $\mathcal{L}(\text{DCS})$ , if the degree of non-regulation is equal to 1. In general, it is interesting to do further research on those programmed grammars with regulation, that is with degree of non-regulation 1, and their families of languages.

Generating programmed grammars  $G$  with  $Rg(G) = 1$ , except for grammars with non-context-free rules *and* with appearance checking, can generate finite languages only.

**Lemma 3.50** ([21]) The following families of languages equal the family of all finite languages:

$$\begin{aligned} \mathcal{L}_1^{\text{gen}}(\text{P[-left]}, \text{REG}) &= \mathcal{L}_1^{\text{gen}}(\text{P[-left]}, \text{REG}, \text{ac}) \\ &= \mathcal{L}_1^{\text{gen}}(\text{P[-left]}, \text{CF}[-\lambda]) = \mathcal{L}_1^{\text{gen}}(\text{P[-left]}, \text{CF}[-\lambda], \text{ac}) \\ &= \mathcal{L}_1^{\text{gen}}(\text{P[-left]}, \text{CS}) = \mathcal{L}_1^{\text{gen}}(\text{P[-left]}, \text{MON}) \\ &= \mathcal{L}_1^{\text{gen}}(\text{P[-left]}, \text{RE}) \end{aligned}$$

**Proof.** Let  $G$  be any programmed grammar of the above mentioned type. If  $L_{\text{gen}}(G) = \emptyset$ , then the statement trivially holds. Now let  $L_{\text{gen}}(G) \neq \emptyset$  and  $w \in L_{\text{gen}}(G)$  be obtained with exactly  $n$  derivation steps. By induction one can show that all sentential forms which can be derived from the axiom by a fixed number of derivation steps starting off with one and the same production applied to the axiom are equally long and have the same Parikh vector. Hence all words which can be obtained with  $n$  derivation steps are letter equivalent to the word  $w$ , thus are terminal words.  $\square$

If core rules of type  $X$  with  $X \in \{\text{CS}, \text{MON}, \text{RE}\}$  and appearance checking are allowed, the family  $\mathcal{L}(X)$  is characterized both in generating and in accepting mode, see Lemma 3.42.

Accepting programmed grammars without appearance checking do not lead to the same trivial characterization of the class of finite languages as in Lemma 3.50. For example, even

a right-linear programmed grammar with regulation 1 but *without* left-most derivations and *without* appearance checking can accept a regular and non-finite language. This is seen as follows: Let  $G = (\{S\}, \{a\}, P, S)$ , where  $P$  contains the productions

$$\begin{aligned} (r_1 : a \rightarrow S, \{r_2\}, \emptyset), \\ (r_2 : aS \rightarrow S, \{r_2\}, \emptyset). \end{aligned}$$

Then it is easy to see that  $L_{\text{acc}}(G) = a^+$ . Therefore the language  $a^+$  belongs to  $\mathcal{L}_1^{\text{acc}}(\text{P}, \text{REG})$ . By a symmetric argument one can use also left-linear or context-free core rules, thus covering also the left-most derivation case.

Concerning accepting programmed grammars with context-free core rules and appearance checking and with regulation, we only mention the following facts demonstrating that this class is a non-trivial one, even if  $\lambda$ -rules are forbidden.

For example, the (non-deterministic) context-free language  $L = \{ww^R \mid w \in \{a, b\}^*\}$ , where  $w^R$  denotes the mirror image of  $w$ , is contained in  $\mathcal{L}_1^{\text{acc}}(\text{P-left}, \text{CF}-\lambda, \text{ac})$ . This can be seen as follows. Consider the accepting context-free  $\lambda$ -free programmed grammar  $G = (V_N, V_T, P, S)$  working in leftmost manner with  $V_T = \{a_1, a_2, \dots, a_k\}$ , which first guesses the symbols that form the left and the right margin of the terminal word to be analyzed by a production of the form

$$([L, i, j] : a_i \rightarrow a_i^L, [i, j, 1], \emptyset),$$

for  $1 \leq i, j \leq k$ . Now, the leftmost occurrence of symbol  $a_i$  is marked by superscript  $L$  and  $G$  starts to search for the rightmost occurrence of symbol  $a_j$  and marks it by superscript  $R$  with help of the following group of productions:

$$\begin{aligned} ([i, j, 1] : a_j \rightarrow a'_j, [i, j, 2], \emptyset), \\ ([i, j, 2] : a_j \rightarrow a'_j, [i, j, 3], [i, j, 4]), \\ ([i, j, 3] : a'_j \rightarrow a''_j, [i, j, 2], \emptyset), \\ ([i, j, 4] : a'_j \rightarrow a_j^R, \sigma([i, j, 4]), \emptyset). \end{aligned}$$

If the guess was correct, the margin symbols are marked by superscripts  $L$  and  $R$ , and if these margin symbols consume the string in an appropriate way,  $G$  can check whether or not the input string does belong to  $L$ .

Moreover, it is easily shown that the family  $\mathcal{L}_1^{\text{acc}}(\text{P-left}, \text{CF}-\lambda, \text{ac})$  contains even non-context-free languages like  $\{a^{2^n} \mid n \geq 0\}$  or  $\{w cw \mid w \in \{a, b\}^*\}$ , it includes all regular languages, and that it is closed under union.

The same properties can be shown for the class  $\mathcal{L}_1^{\text{acc}}(\text{P}, \text{CF}-\lambda, \text{ac})$ ; note that, in the first example, the margin symbols can directly be marked in the beginning of any derivation by non-deterministic choice.

### 3.3 Discussion

The descriptive power of accepting grammars has been studied in detail. Accepting grammars can be viewed as nondeterministic recognizers for the language families which they describe. In many cases, when in any possible derivation the lengths of the sentential forms build a monotone decreasing sequence of non-negative integers, then backtrack parsing algorithms may be obtained from the accepting grammar formalisms.

Unfortunately, most of those mechanisms lose too many positive properties of the context-free grammars, in particular the fixed membership problem becomes intractable. If the family of context-sensitive languages is effectively characterized, then this decidability problem is **PSPACE**-complete; if accepting grammars are constructively equivalent to their generating variants, then the problem is also at least **NP**-hard in many cases. Even the known deterministic polynomial-time parsing algorithms for tree-adjoining (and equivalent mildly context-sensitive grammars) are of complexity  $O(n^6)$ , see, for example, [67].

In the last section of this chapter, another use of accepting grammars has been taken into consideration: we presented several *grammatical* characterizations of the LBA problem. Thus, a problem stemming from complexity theory can be expressed in terms of descriptive complexity of grammars. Alas, for the precise characterization of the LBA problem we needed context-sensitive core rules. It is an open problem whether or not the inclusion  $\mathcal{L}_1^{\text{acc}}(\text{P-left}, \text{CF}-\lambda, \text{ac}) \subseteq \mathcal{L}(\text{DCS})$  is strict.

The exact characterizations of the LBA problem can be summarized as follows:

$$\mathcal{L}_1^{\text{acc}}(\text{P-left}, \text{CS}, \text{ac}) = \mathcal{L}(\text{DCS})$$

and

$$\mathcal{L}_1^{\text{acc}}(\text{P}, \text{CS}, \text{ac}) = \mathcal{L}_2^{\text{acc}}(\text{P-left}, \text{CS}, \text{ac}) = \mathcal{L}(\text{CS}).$$

Thus, giving up determinism in linear bounded automata means the same as giving up either the restriction to leftmost derivations or the restriction to a degree of non-regulation of 1 in accepting programmed grammars with context-sensitive core rules. Concerning future work, it seems to be interesting to further explore the degree of non-regulation, for example, for (programmed) grammars with timed bounded functions.

Let us remark that by the technique given in the previous section for finding the margins of a given terminal word, that is, the rules of group (2) in the proof of Lemma 3.46 can be replaced with productions having context-free  $\lambda$ -free core rules. Thus, monotone rules are in fact only needed in the productions of groups (4) and (5) in that proof. Moreover, it is remarkable that we actually do not need the left-most derivation feature in the proof of Lemma 3.46, what reprovves that, for  $X \in \{\text{CS}, \text{MON}\}$ , we have

$$\mathcal{L}(\text{DCS}) = \mathcal{L}_1^{\text{acc}}(\text{P-left}, X, \text{ac}) \subseteq \mathcal{L}_1^{\text{acc}}(\text{P}, X, \text{ac}) = \mathcal{L}(\text{CS})$$

what has already been shown because of Lemma 3.42.

As it has been mentioned in Section 3.1.1, also *pure* grammars and systems might be taken into consideration, what is of some interest with respect to syntactical analysis issues. What is the general idea of the concept of a pure grammar? Take the definition of successful derivation of the non-pure case, but do not distinguish between terminal and nonterminal symbols. (This means especially that terminal symbols may be rewritten in generating context-free grammars, and they can be generated in accepting context-free grammars.) Into the described pure language, we put all words appearing in some successful derivation. Such pure grammars have been investigated, e.g., in [57, 72], and in combination with regulated rewriting in [36, 40], and they are of interest because of the following reasons:

1. For the simulation of say type-0 grammars by accepting programmed grammars, a suitable nonterminal alphabet is needed. Pure grammars are a suitable mean in order to get rid of the influence of coding tricks possible by making use of a nonterminal alphabet.



2. The “intermediate” sentential forms belong to the language such that no information about the derivation process is lost. This is important, for example, for purposes of syntax analysis. Moreover, new light is shed on the role of the auxiliary symbols in the simulation of generating grammars by accepting ones: Is it possible to keep the simulation features also in the pure case or do we find new relationships between the families of languages generated and accepted by certain pure grammars?
3. In some sense, they form a sequential counterpart to the basic (non-extended) types of Lindenmayer systems, where, in the terminology of Definition 2.7,  $\Delta = \Sigma$ .

Accepting pure grammars and systems have been investigated in [18] and [19]. As in the non-pure case, both equivalences of the generating and the accepting variants of grammar formalisms and strict inclusions of the generated in the accepted families of languages have been obtained. Additionally, also examples of pure grammars and systems have been encountered for which incomparability between the generating and the accepting modes are proved.

Concerning the yield relation, we textually transferred the given definition of a derivation step in the generating device to the accepting one. One could alternatively try to create an accepting grammar type which mimics the generating process of another generative grammar step by step. Such an approach has been taken in [76] for the case of cooperating distributed grammar systems. By construction, the language families generated and accepted in this way automatically coincide. We pursued that approach here only for PC grammar systems, where it led to non-straightforward considerations.

## Chapter 4

# Leftmost Derivations

For context-free grammars, the theory of parsing is sophisticated and methodologically sound. One fundamental property which many compiler generations make use of is the fact that every context-free grammar still describes the same language if it is restricted to left-most derivations, that is, if at any step of the derivation the leftmost occurrence of a nonterminal symbol is replaced. In the case of context-free grammars with controlled derivations, such as context-free matrix or context-free programmed grammars, the situation is quite different, see for example [41]: at first, the descriptive capacity of the grammars may change and, secondly, there are frequently several different definitions of the leftmost restriction, which are all sensible and may yield different language families.

In the preceding chapter, in Section 3.2, programmed grammars have been restricted to leftmost derivations in order to eliminate one sort of nondeterminism in the derivations. The results from [41] have been supplemented there, taking also programmed grammars with regular, context-sensitive, monotone and type-0 core rules into consideration. Moreover, besides generating grammars also the accepting case has been regarded.

For context-free matrix grammars without appearance checking, Dassow, Fernau and Păun systematically investigated the concept of leftmost derivation in [38], where twelve different definitions of leftmostness have been considered. Seven of them have been shown to characterize the set of all recursively enumerable languages. Precise characterizations of the families of context-free and context-sensitive languages have also been obtained. In three cases, the closure under a homomorphism and an intersection with a regular language yields characterizations of the family  $\mathcal{L}(\text{RE})$ . These investigations have been expanded to further classes of grammars with controlled derivations in [51]. There, grammars controlled by a bicoloured digraph viewed as a unifying generalization of several types of grammars with controlled derivations are in the focus of the investigations.

This chapter deals with CD grammar systems with respect to the leftmost feature. Similarly to the case of matrix grammars, several natural restrictions are applicable. The investigation of two of them has already been started by Dassow and Mitrană, see [39]. There, one distinguishes between a *strong* type of leftmostness, where always the leftmost occurrence of a symbol has to be rewritten, and a *weak* leftmost feature, where the active context-free grammar of the system (that is, the component which is currently working on the sentential form) has to rewrite the leftmost nonterminal which it is able to rewrite. But also in this weak type of leftmostness any component has to replace the very leftmost occurrence of a nonterminal (according to the strong leftmost definition) in its first step. That is, the system

is forced to treat the leftmost nonterminal when no component has been activated yet, and then the leftmost nonterminals for which the activated component has a rule.

As this type, which will be called sw-type in the present chapter, is the hardest (“as leftmost as possible”) way of leftmost rewriting in which cooperating distributed grammar systems are able to generate non-context-free languages, it will be used in the next chapter (following [24]) for defining the grammar class having a deterministic parser of strictly sub-quadratic time complexity.

But one could also apply the weak manner of leftmostness in the first steps to be performed by the components or the strong one only in the following steps. Taking also the free type of rewriting, that is, the ordinary “non-leftmost” replacement into consideration, one can distinguish nine ways in which the components work on the sentential forms.

This chapter aims to provide a systematic investigation of the power of the corresponding CD grammar systems, working according to several different cooperation strategies. Although there is a close relationship between matrix grammars and CD grammar systems, the results and the proof techniques presented in [38] and the present chapter differ considerably. This might mainly be due to the fact that in matrix grammars the sequence of rules (in the matrices) is prescribed and cannot be changed.

## 4.1 Definitions

First, a restricted type of programmed grammars is defined, which has been introduced in [106] as another characterization of the language family generated by restricted ETOL systems, where the applicability of the tables is controlled by random context conditions.

**Definition 4.1** A *context-free recurrent programmed grammar* [106] is a context-free programmed grammar where, for each of its production rules  $(r : A \rightarrow z, \sigma(r), \phi(r))$ , (1) the label  $r$  is contained in the success field,  $r \in \sigma(r)$ , and (2) either  $\phi(r) = \sigma(r)$  or  $\phi(r) = \emptyset$ .

Like a programmed grammar as defined in 2.2, a recurrent programmed grammar is said to be *without appearance checking* if the failure field is empty for each production in  $P$ . In the notation of language families, we write rP instead of P whenever programmed grammars are restricted to recurrent programmed grammars. The following inclusion chains hold due to [44, 50, 106].

$$\begin{aligned} \mathcal{L}(\text{CF}) &\subset \mathcal{L}(\text{rP}, \text{CF}) \subseteq \mathcal{L}(\text{P}, \text{CF}) \subset \mathcal{L}(\text{P}, \text{CF}, \text{ac}) = \mathcal{L}(\text{RE}). \\ \mathcal{L}(\text{CF}) &\subset \mathcal{L}(\text{rP}, \text{CF}) \subset \mathcal{L}(\text{rP}, \text{CF}, \text{ac}) \subseteq \mathcal{L}(\text{P}, \text{CF}, \text{ac}) = \mathcal{L}(\text{RE}). \end{aligned}$$

Next we turn to the definition of the various leftmost restricted variants of context-free CD grammar systems.

**Definition 4.2** ([39, 23, 25]) Let  $\Gamma = (N, T, S, P_1, P_2, \dots, P_n)$  be a CD grammar system with  $n$  components.

A *strong leftmost rewriting step* is a direct derivation step that rewrites the leftmost nonterminal of a sentential form. Formally, we write  $x \xRightarrow{s}_i y$  if and only if  $x = x_1 A x_2$ ,  $y = x_1 z x_2$ ,  $A \rightarrow z \in P_i$ , and  $x_1 \in T^*$ ,  $x_2, z \in (N \cup T)^*$ .

A *weak leftmost rewriting step* is a direct derivation step that rewrites the leftmost non-terminal from the domain of the active component. Formally,  $x \xRightarrow{w}_i y$  where  $x = x_1 A x_2$ ,  $y = x_1 z x_2$ ,  $A \rightarrow z \in P_i$ , and  $x_1 \in T_i^*$ ,  $x_2, z \in (N \cup T)^*$ .

A *free rewriting step* is a usual direct derivation step that rewrites any nonterminal of a sentential form. Formally,  $x \xrightarrow{f}_i y$  where  $x = x_1 A x_2$ ,  $y = x_1 z x_2$ ,  $A \rightarrow z \in P_i$ , and  $x_1, x_2, z \in (N \cup T)^*$ .

Let  $\xrightarrow{\alpha}_i^k$  denote  $k$  consecutive derivation steps of component  $P_i$ ,  $k \geq 1$ , in the strong leftmost, weak leftmost or free manner, if  $\alpha = s$ ,  $\alpha = w$ , or  $\alpha = f$ , respectively. Further, let  $\xrightarrow{\alpha}_i^*$  denote the reflexive and transitive closure of  $\xrightarrow{\alpha}_i$ .

We denote the combinations of the above defined types of leftmostness as  $\alpha\beta$  where  $\alpha, \beta \in \{s, w, f\}$ . The first symbol,  $\alpha$ , denotes the type of leftmostness applied when a new component is chosen to begin the rewriting, that is, if  $\alpha = s$  then the chosen component must be able to rewrite the leftmost nonterminal of the sentential form and it must rewrite this nonterminal in the first rewriting step; if  $\alpha \in \{w, f\}$ , then the chosen component must be able to rewrite some of the nonterminals in the sentential form, and in case of  $\alpha = w$ , the leftmost occurrence of these nonterminals must be rewritten. The symbol  $\beta$  denotes the restriction obeyed by the chosen component for the rewriting steps following the first one.

**Definition 4.3** ([23, 25]) Let  $\Gamma = (N, T, S, P_1, P_2, \dots, P_n)$  be a CD grammar system with  $n$  context-free components. A *\*-derivation* by the  $i$ th component in the  $\alpha\beta$  manner, where  $\alpha, \beta \in \{s, w, f\}$ , denoted by  $\xrightarrow{\alpha\beta}_i^*$ , is defined as

$$x \xrightarrow{\alpha\beta}_i^* y, \text{ iff } x \xrightarrow{\alpha}_i x_1 \text{ and } x_1 \xrightarrow{\beta}_i^* y,$$

for some  $x_1 \in (N \cup T)^*$ . A *t-derivation* by the  $i$ th component in the  $\alpha\beta$  manner, where  $\alpha, \beta \in \{s, w, f\}$ , denoted by  $\xrightarrow{\alpha\beta}_i^t$ , is defined as

$$x \xrightarrow{\alpha\beta}_i^t y, \text{ iff } x \xrightarrow{\alpha}_i x_1, x_1 \xrightarrow{\beta}_i^* y, \text{ and there is no } z \text{ with } y \xrightarrow{\beta}_i z,$$

for some  $x_1 \in (N \cup T)^*$ . Let  $k$  be a positive integer. A *k-step derivation* by the  $i$ th component in the  $\alpha\beta$  manner, where  $\alpha, \beta \in \{s, w, f\}$ , denoted by  $\xrightarrow{\alpha\beta}_i^{\leq k}$ , is defined as

$$x \xrightarrow{\alpha\beta}_i^{\leq k} y, \text{ iff } x \xrightarrow{\alpha}_i x_1 \text{ and } x_1 \xrightarrow{\beta}_i^{k-1} y,$$

for some  $x_1 \in (N \cup T)^*$ . An *at most k-step (at least k-step) derivation* by the  $i$ th component in the  $\alpha\beta$  manner, where  $\alpha, \beta \in \{s, w, f\}$ , denoted by  $\xrightarrow{\alpha\beta}_i^{\leq k}$  ( $\xrightarrow{\alpha\beta}_i^{\geq k}$ ), is defined as

$$x \xrightarrow{\alpha\beta}_i^{\leq k} y \text{ (} x \xrightarrow{\alpha\beta}_i^{\geq k} y \text{), iff } x \xrightarrow{\alpha\beta}_i^{\leq k'} y \text{ for some } k' \leq k \text{ (} k' \geq k \text{)}.$$

A *derivation in the full-competence mode* by the  $i$ th component in the  $\alpha\beta$  manner, where  $\alpha, \beta \in \{s, w, f\}$ , denoted by  $\xrightarrow{\alpha\beta}_i^{\text{full}}$ , is defined as

$$x \xrightarrow{\alpha\beta}_i^{\text{full}} y, \text{ iff } x = x_0 \xrightarrow{\alpha}_i x_1 \xrightarrow{\beta}_i x_2 \xrightarrow{\beta}_i \dots \xrightarrow{\beta}_i x_m = y,$$

where for all  $j$ ,  $0 \leq j \leq m - 1$ , we have

$$x_j \in (\text{dom}(P_i) \cup T)^*, \text{ and } \text{symb}(y) \cap (N \setminus \text{dom}(P_i)) \neq \emptyset, \text{ or } y \in T^*.$$

The “non-leftmost” way of rewriting (or the ff manner in the above notation) is exactly the same as it has been introduced in 2.13.

**Definition 4.4** ([23, 25]) Let  $\Gamma$  be a CD grammar system as above. The language generated by  $\Gamma$  in the  $\alpha\beta$  manner of leftmostness and the mode  $\gamma$  of derivation,  $\alpha, \beta \in \{s, w, f\}$ ,  $\gamma \in \{*, t, \text{full}\} \cup \{=k, \leq k, \geq k \mid k \geq 1\}$ , is

$$L_{\alpha\beta}(\Gamma, \gamma) = \{w \in T^* \mid S \xrightarrow{\alpha\beta}_{i_1} w_1 \xrightarrow{\alpha\beta}_{i_2} w_2 \dots \xrightarrow{\alpha\beta}_{i_m} w_m = w, \\ \text{where } m \geq 1, 1 \leq i_j \leq n, w_j \in (N \cup T)^*, 1 \leq j \leq m\}.$$

In what follows, the notation is simplified by letting  $\mathcal{L}_{\alpha\beta}(\text{CD}_n, \gamma)$  denote the class of languages generated by CD grammar systems with  $n$  components in the  $\alpha\beta$  manner of leftmostness and the mode  $\gamma$  of derivation,  $\alpha, \beta \in \{s, w, f\}$ ,  $\gamma \in \{*, t, \text{full}\} \cup \{=k, \leq k, \geq k \mid k \geq 1\}$ ; correspondingly, let  $\mathcal{L}_{\alpha\beta}(\text{CD}_\infty, \gamma) = \bigcup_{n \geq 1} \mathcal{L}_{\alpha\beta}(\text{CD}_n, \gamma)$ .

Obviously, the ff-type of leftmostness imposes no restriction at all, therefore, we will omit the subscript ff in the sequel without further mentioning. Note also that the ss and the sw variants were called “strong leftmost” and “weak leftmost” in [39].

## 4.2 The Power of Leftmost Derivations in CD Grammar Systems

In the following we will investigate the power of context-free CD grammar systems using the above defined modes of derivation in the leftmost manner.

### 4.2.1 The So-Called Trivial Derivation Modes

The following theorem shows that restricting the derivations of CD grammar systems in the above described combinations of leftmostness does not increase their power if they use the so-called “trivial” derivation modes, that is, the modes which in the “non-leftmost” manner generate context-free languages only (see item 1. of the results above).

**Theorem 4.1** ([23, 25]) For all  $\alpha \in \{ss, sw, sf, ws, ww, wf, fs, fw\}$ ,  $\beta \in \{*, =1, \leq 1, \geq 1\} \cup \{\leq k \mid k \geq 2\}$ ,  $\mathcal{L}_\alpha(\text{CD}_\infty, \beta) = \mathcal{L}(\text{CF})$ .

**Proof.** To show that  $\text{CF} \subseteq \mathcal{L}_\alpha(\text{CD}_\infty, \beta)$ , let  $G = (N, T, S, P)$  be a context-free grammar, and consider the CD grammar system  $\Gamma = (N, T, S, P_1)$  with  $P_1 = P$ . Since a context-free derivation produces the same result irrespective of the order in which the rewritten nonterminals are chosen, the language generated by  $\Gamma$  is the same as the one generated by  $G$ , thus,  $L_\alpha(\Gamma, \beta) = L(G)$ . Note that the grammar system  $\Gamma$  is allowed to change the component after any single derivation step, according to all derivation modes  $\beta$  considered in this theorem.

In order to prove the inclusion  $\mathcal{L}_\alpha(\text{CD}_\infty, \beta) \subseteq \text{CF}$ , consider the CD grammar system  $\Gamma = (N, T, S, P_1, P_2, \dots, P_n)$ , and let  $G = (N, T, S, P)$  be a context-free grammar where  $P = \bigcup_{i=1}^n P_i$ . Since any derivation of  $\Gamma$  in any of the derivation modes  $\beta$  and manners  $\alpha$  of leftmostness can also be reproduced by  $G$ , we have  $L_\alpha(\Gamma, \beta) \subseteq L(G)$ . To prove that  $L(G) \subseteq L_\alpha(\Gamma, \beta)$ , notice that any of the derivation modes  $\beta$  allows to use a component just for one rewriting step, thus, for  $\alpha \in \{fs, fw\}$  arbitrary derivations of  $G$  can be reproduced by  $\Gamma$ , while for  $\alpha \in \{ss, sw, sf, ws, ww, wf\}$  all leftmost derivations of  $G$  can be reproduced by  $\Gamma$ , where by leftmost derivations of  $G$  we mean derivations which always replace the leftmost nonterminal occurrence of the sentential form. Since all words which can be derived by  $G$ , can also be derived by  $G$  in the leftmost manner, we have  $L(G) \subseteq L_\alpha(\Gamma, \beta)$ , and thus,  $L(G) = L_\alpha(\Gamma, \beta)$ .  $\square$

### 4.2.2 The Terminal Mode of Derivation

Let us now continue with the study of the  $t$  mode. By Lemma 2 of [39], we have

$$\mathcal{L}_{ss}(\text{CD}_\infty, t) = \mathcal{L}(\text{CF}).$$

Concerning the other types of leftmostness, context-free CD grammar systems working in the  $t$ -mode of derivation are equally powerful when working in the  $sw$  and  $sf$  manner. To see this observe that, given a sentential form, (a) the same components can be chosen to rewrite since the leftmost nonterminal has to be rewritten in both cases, and (b) the chosen components must give the same results in the  $sw$  and the  $sf$  manner because of the  $t$ -mode. Thus, according to Lemma 5 of [39] we have, for  $\alpha \in \{sw, sf\}$ ,

1.  $\mathcal{L}(\text{CF}) = \mathcal{L}_\alpha(\text{CD}_1, t) = \mathcal{L}_\alpha(\text{CD}_2, t)$  and
2.  $\mathcal{L}(\text{ET0L}) \subset \mathcal{L}_\alpha(\text{CD}_3, t) = \mathcal{L}_\alpha(\text{CD}_\infty, t)$ .

Based on similar arguments taking into account the nature of the  $t$ -mode derivations, the following three variants of leftmostness have no effect on the generative power of CD grammar systems.

**Theorem 4.2 ([23, 25])** For  $\alpha \in \{ww, wf, fw\}$ ,  $n \geq 1$ ,  $\mathcal{L}_\alpha(\text{CD}_n, t) = \mathcal{L}(\text{CD}_n, t)$ .

**Proof.** First, given any CD grammar system  $\Gamma$  and a sentential form  $w$ , a component can start to rewrite  $w$  using one of the  $ww, wf, fw$  leftmost restrictions if and only if it can also start to rewrite  $w$  in the non-leftmost manner. Second, since the components must rewrite the sentential form as long as it contains at least one nonterminal from the domain of the component, the result of the rewriting process is the same in the non-leftmost manner, or using any of the  $ww, wf, fw$  leftmost restrictions.  $\square$

Concerning the  $ws$  and  $fs$  variants of leftmostness, the argument above does not work, but we can show the following weaker statement.

**Theorem 4.3 ([23, 25])** For  $n \geq 5$ , and  $\alpha \in \{ws, fs\}$ ,  $\mathcal{L}(\text{CF}) \subset \mathcal{L}_\alpha(\text{CD}_n, t)$ .

**Proof.** The inclusion follows from the fact that any context-free CD grammar system with only one component is just a context-free grammar, and it generates the same context-free language when working in the  $t$ -mode of derivation. For the strictness we argue as follows. Consider the CD grammar system

$$\Gamma = (\{S, A, A', B, B', T, F\}, \{a, b, c\}, S, P_1, P_2, \dots, P_5),$$

a CD grammar system with five components, where

$$\begin{aligned} P_1 &= \{S \rightarrow AB, A' \rightarrow A\}, \\ P_2 &= \{A' \rightarrow F, B' \rightarrow B, B' \rightarrow T\}, \\ P_3 &= \{A \rightarrow aA'b\}, \\ P_4 &= \{A \rightarrow \varepsilon, T \rightarrow \varepsilon, B \rightarrow F, B' \rightarrow F\}, \\ P_5 &= \{A \rightarrow F, B \rightarrow cB'\}. \end{aligned}$$

This system generates a non-context-free language, namely, if  $\alpha \in \{ws, fs\}$ , then  $L_\alpha(\Gamma, t) = \{a^i b^i c^j \mid 1 \leq j \leq i\}$ .  $\square$

### 4.2.3 The $=k$ and $\geq k$ Modes of Derivation

The investigation of leftmostness for the derivation modes of  $=k$  and  $\geq k$  for  $k \geq 2$  has also been started in [39]. By Lemma 3, Lemma 4, and Corollary 1 of [39], we have for  $n \geq 1$  and  $k \geq 1$ ,

1.  $\mathcal{L}_{\text{ss}}(\text{CD}_n, =k) = \mathcal{L}_{\text{ss}}(\text{CD}_n, \geq k) = \mathcal{L}(\text{CF})$ ,
2.  $\mathcal{L}_{\text{sw}}(\text{CD}_n, =k) \subseteq \mathcal{L}_{\text{sw}}(\text{CD}_n, =p \cdot k)$  where  $p \geq 1$ , and  
 $\mathcal{L}_{\text{sw}}(\text{CD}_n, \leq k) \subseteq \mathcal{L}_{\text{sw}}(\text{CD}_{nk}, =q)$  where  $q$  is the least common multiple of  $1, 2, \dots, k$ .

Note that the second part of item 2 is also a consequence of Theorem 4.1 above. Note also, that item 1 of the above statement implies that, for  $k \geq 2$  and  $n \geq 1$ ,

$$\mathcal{L}_{\text{ws}}(\text{CD}_n, =k) = \mathcal{L}_{\text{ws}}(\text{CD}_n, \geq k) = \mathcal{L}(\text{CF}).$$

To see this, observe that if a component of a CD grammar system is able to perform at least two derivation steps in the ws manner on a sentential form, then it must be able to rewrite the leftmost nonterminal of the sentential form, thus any weak leftmost rewriting step must also be strong leftmost, that is, for  $k \geq 2$ ,  $\text{ws} = \text{ss}$  holds.

Now we continue the investigations concerning the  $=k$  and  $\geq k$  modes with the different types of leftmost restrictions. In contrast to item 2 above, we can show the following.

**Theorem 4.4 ([23, 25])** For any  $n, k \geq 1$  and  $\alpha \in \{\text{sw}, \text{ww}\}$ ,

1.  $\mathcal{L}_{\text{ww}}(\text{CD}_n, =k) \subseteq \mathcal{L}_\alpha(\text{CD}_{n+2}, =k+1)$ , and
2.  $\mathcal{L}_{\text{ww}}(\text{CD}_n, \geq k) \subseteq \mathcal{L}_\alpha(\text{CD}_{n+2}, \geq k+1)$ .

**Proof.** Let  $\Gamma = (N, T, S, P_1, \dots, P_n)$  be a CD grammar system and let  $k \geq 1$  be a given integer. We construct a system  $\Gamma'$  with  $n+2$  components such that  $L_{\text{ww}}(\Gamma, =k) = L_\alpha(\Gamma', =k+1)$  and  $L_{\text{ww}}(\Gamma, \geq k) = L_\alpha(\Gamma', \geq k+1)$ ,  $\alpha$  as above.

Let  $\Gamma' = (N', T, S', P'_1, P'_2, \dots, P'_n, P_{\text{init+reset}}, P_{\text{ter}})$  with

$$N' = N \cup \{S', X', Y\} \cup \{X_i \mid 0 \leq i \leq k\},$$

the unions being pairwise disjoint, and let the components be defined as follows. The derivations start with the rules  $S' \rightarrow X_0SY$  and  $X_0 \rightarrow X_0$  of the component

$$P_{\text{init+reset}} = \{S' \rightarrow X_0SY, X_0 \rightarrow X_0\} \cup \{X' \rightarrow X_1, X_k \rightarrow X_0\} \cup R_{\text{reset}},$$

where

$$R_{\text{reset}} = \begin{cases} \{X_i \rightarrow X_{i+1} \mid 1 \leq i \leq k-1\} & \text{if } k \geq 2, \\ \emptyset & \text{if } k = 1. \end{cases}$$

Now, every sentential form  $w$  of  $\Gamma$  corresponds to the sentential form  $X_0wY$  of  $\Gamma'$ . To simulate a  $k$ -step application of the component  $P_i$  with  $k+1$  steps, we have for each  $i$ ,  $1 \leq i \leq n$ ,

$$P'_i = P_i \cup \{X_0 \rightarrow X'\}.$$

With these components we obtain the sentential form  $X'w'Y$ , if and only if  $w'$  can be obtained by the corresponding component of  $\Gamma$ . Note that  $X_0$  must be rewritten due to the leftmost restriction.

To reset  $X'$  to  $X_0$  again, we use the the rules of  $P_{\text{init+reset}}$ , and for the termination of the derivation we need

$$P_{\text{ter}} = \{ Y \rightarrow \varepsilon, A \rightarrow F \mid A \in N \} \cup R_{\text{ter}},$$

where

$$R_{\text{ter}} = \begin{cases} \{X_0 \rightarrow \varepsilon\} & \text{if } k = 1, \\ \{X_i \rightarrow X_{i+1} \mid 0 \leq i \leq k-2\} \cup \{X_{k-1} \rightarrow \varepsilon\} & \text{if } k \geq 2. \end{cases}$$

Since any  $k$ -step derivation is also a  $\geq k$ -step derivation, the construction above is sufficient to prove both statements of our theorem.  $\square$

We can also show that CD grammar systems using the sf leftmost restriction in the  $\geq(k+1)$  derivation modes are at least as powerful as non-restricted systems working in the  $\geq k$  mode.

**Theorem 4.5 ([23, 25])** For any  $n, k \geq 1$ ,  $\mathcal{L}(\text{CD}_n, \geq k) \subseteq \mathcal{L}_{\text{sf}}(\text{CD}_{n+1}, \geq k+1)$ .

**Proof.** Let  $\Gamma = (N, T, S, P_1, P_2, \dots, P_n)$  be a CD grammar system and let  $k \geq 1$  be a given integer. We construct a system  $\Gamma'$  with  $n+1$  components such that  $L(\Gamma, \geq k) = L_{\text{sf}}(\Gamma', \geq k+1)$ .

Let  $\Gamma' = (N \cup \{S', X, X'\}, T, S', P_0, P'_1, \dots, P'_n)$  with

$$P_0 = \{S' \rightarrow S', S' \rightarrow XS, X' \rightarrow X', X' \rightarrow X, X' \rightarrow \varepsilon\},$$

and

$$P'_i = P_i \cup \{X \rightarrow X'\} \text{ for } 1 \leq i \leq n.$$

The work of  $\Gamma'$  starts with the possibly repeated application of  $P_0$  which yields the sentential form  $XS$ . Now, since  $X$  is the leftmost nonterminal, any  $\geq(k+1)$  step application of a component with the sf leftmost restriction to a sentential form  $Xw$  leads to  $X'w'$  where  $w'$  is the result of the free  $\geq k$  step application of the same component to the sentential form  $w$ . If we use  $P_0$  to change  $X'$  back to  $X$ , we can repeat the simulation of any  $\geq k$  step of  $\Gamma$  in  $\Gamma'$  until  $X'$  is erased and a terminal string is produced. If after the removal of  $X'$  from the sentential form,  $\Gamma'$  does not produce a terminal word but continues to apply its components in the  $\geq(k+1)$  mode, then it executes special  $\geq k$  step derivations, so it cannot produce any word which cannot be also produced by  $\Gamma$  in the  $\geq k$  mode. Thus the two systems generate the same language, and the statement is proved.  $\square$

Note that the argument above cannot be used to prove a similar statement for the  $=k$  mode of derivation, since  $=k$  step derivations cannot be viewed as special  $=k$  step derivations, contrarily to the cases of  $\geq k$  and  $\geq(k+1)$  step derivations, for  $k \geq 1$ .

Now we continue with the characterization of the generative power of systems using either the  $=k$  or the  $\geq k$  derivation modes for  $k \geq 2$ . First we show that systems using the sf restriction can generate non-context-free languages.

**Theorem 4.6 ([23, 25])** For  $n \geq 4$  and  $\beta \in \{=k, \geq k \mid k \geq 2\}$ ,  $\mathcal{L}(\text{CF}) \subset \mathcal{L}_{\text{sf}}(\text{CD}_n, \beta)$ .

**Proof.** The inclusion follows from the following observation. Given a context-free grammar  $G = (N, T, S, P)$ , let  $X$  be a new nonterminal symbol, and replace  $P$  with

$$P \cup \{S \rightarrow XS, X \rightarrow X, X \rightarrow \varepsilon\}.$$

Now, we interpret  $G$  as CD grammar system with one component. In the first, strong leftmost step, the component must replace the symbol  $X$ , and then simulate the actual context-free



derivation step. As  $X$  can be deleted in the very last step, all derivations of  $G$  can be simulated. If  $X$  has been erased when other nonterminals remained in the sentential form, then only derivations are possible which can be performed in the given context-free grammar, either. Hence, the language  $L(G)$  is generated.

To prove the strictness, we use a similar construction as in the proof of Theorem 4.3. Consider the CD grammar system with four components

$$\Gamma = (\{S, A, A', B, B', T\}, \{a, b, c\}, S, P_1, P_2, P_3, P_4),$$

where

$$\begin{aligned} P_1 &= \{S \rightarrow AB, A \rightarrow A, T \rightarrow T, T \rightarrow \varepsilon\}, \\ P_2 &= \{A' \rightarrow A, B' \rightarrow B, B \rightarrow B\}, \\ P_3 &= \{A \rightarrow aA'b, B \rightarrow cB', B' \rightarrow B'\}, \\ P_4 &= \{A \rightarrow ab, B \rightarrow cT, T \rightarrow T\}. \end{aligned}$$

This system generates a non-context-free language, for  $\beta \in \{=k, \geq k \mid k \geq 2\}$ ,  $L_{\text{sf}}(\Gamma, \beta) = \{a^i b^i c^i \mid i \geq 1\}$ .  $\square$

The next theorem characterizes the power of the sw, ww, and fw types of leftmostness. We give a more detailed proof since the other theorems of this section are proved by modifications of this construction.

**Theorem 4.7** ([23, 25]) For  $\alpha \in \{\text{sw}, \text{ww}, \text{fw}\}$ , and  $k \geq 2$ ,

$$\mathcal{L}_\alpha(\text{CD}_\infty, =k) = \mathcal{L}_\alpha(\text{CD}_\infty, \geq k) = \mathcal{L}(\text{RE}).$$

**Proof.** For any recursively enumerable language  $L$ , there is a context-free programmed grammar  $G = (N, T, S, P)$  with appearance checking such that  $L = L_{\text{left}}(G)$ , where  $L_{\text{left}}(G)$  denotes the language generated by  $G$  in such a way that each rule is always applied to rewrite the leftmost occurrence of the nonterminal on its left-hand side. According to the proof Lemma 1.4.8 of [41] (where this leftmost derivation mode for programmed grammars is called *left-3*), such a programmed grammar always exists.

Consider the CD grammar system  $\Gamma$  with nonterminal alphabet  $N'$ , terminal alphabet  $T$ , and start symbol  $S_0 \in N'$  where

$$\begin{aligned} N' &= N \cup \{X, X', B, F\} \\ &\cup \{S_i, X_r, X'_r, [r]_i, [r]'_i, (r)_i \mid r \in \text{lab}(P), 0 \leq i \leq k-1\}, \end{aligned}$$

the unions being disjoint, and the components are defined as follows.

$$P_0 = \{S_i \rightarrow S_{i+1} \mid 0 \leq i \leq k-2\} \cup \{S_{k-1} \rightarrow [r]_0 SX \mid r \in \text{lab}(P)\}$$

is the component initiating the derivation process which is continued by simulating the rule labelled with  $r \in \text{lab}(P)$ . The rules of  $G$  are simulated in several rewriting steps.

In order to simulate the successful application of the rules with the sw leftmost restriction, we have a component for each  $(r : A \rightarrow \alpha, \sigma(r), \phi(r))$  defined as

$$\begin{aligned} P_{r, \text{succ}} &= \{[r]_i \rightarrow [r]_{i+1} \mid 0 \leq i \leq k-3\} \\ &\cup \{[r]_{k-2} \rightarrow [r]'_0, A \rightarrow B\alpha, B \rightarrow B\}, \end{aligned}$$

and one more component

$$P_{\text{succ}} = \{ [r]'_i \rightarrow [r]'_{i+1}, [r]'_{k-2} \rightarrow [s]_0 \mid r \in \text{lab}(P), s \in \sigma(r), \\ 0 \leq i \leq k-3 \} \cup \{ B \rightarrow \varepsilon \}.$$

Note that the symbol  $B$  is introduced and deleted in  $P_{r,\text{succ}}$  and  $P_{\text{succ}}$ , respectively, since, in the  $\geq k$ -mode, a multiple application of the rule  $A \rightarrow \alpha$  in one simulation cycle must be prevented.

For the same simulation using the  $\text{ww}$  and  $\text{fw}$  restrictions, we need to make sure that the leftmost marker is rewritten. So for the simulation of the successful rule application we have in this case

$$P'_{r,\text{succ}} = P_{r,\text{succ}} \cup \{ [p]_0 \rightarrow F, [q]'_0 \rightarrow F \mid p, q \in \text{lab}(P), p \neq r \} \\ \cup \{ (p)_0 \rightarrow F \mid p \in \text{lab}(P) \}.$$

To simulate the “unsuccessful” application of the rules with the  $\text{sw}$  restriction, we have for each  $r$  as above

$$P_{r,\text{fail}} = \{ [r]_i \rightarrow [r]_{i+1} \mid 0 \leq i \leq k-3 \} \\ \cup \{ [r]_{k-2} \rightarrow (r)_0, A \rightarrow F, X \rightarrow X \},$$

and one more component

$$P_{\text{fail}} = \{ (r)_i \rightarrow (r)_{i+1}, (r)_{k-2} \rightarrow [s]_0 \mid r \in \text{lab}(P), s \in \phi(r), \\ 0 \leq i \leq k-3 \} \cup \{ X \rightarrow X \}.$$

Note that after reaching the sentential form  $(r)_0 w X$ , where  $w$  corresponds to a sentential form of  $G$ , (at least) one further step has to be performed. If  $A \in \text{symb}(w)$ , then the trap symbol  $F$  will be introduced, blocking the derivation.

For the  $\text{ww}$  and  $\text{fw}$  restrictions we need to replace the component  $P_{r,\text{fail}}$  with two components

$$P'_{r,\text{fail}} = \{ [r]_i \rightarrow [r]_{i+1} \mid 0 \leq i \leq k-3 \} \cup \{ [r]_{k-2} \rightarrow (r)_0, X \rightarrow X_r \}$$

and

$$P''_{r,\text{fail}} = \{ X_r \rightarrow X'_r, X'_r \rightarrow X'_r, A \rightarrow F \},$$

and need to change the rule  $X \rightarrow X$  in  $P_{\text{fail}}$  to  $X'_r \rightarrow X$ .

If a terminal string has been obtained between the left and right markers, then the derivation can be finished by erasing these markers. This can be done with the  $\text{sw}$  or  $\text{ww}$  restriction by the component

$$P_{\text{ter}} = \{ [r]_i \rightarrow [r]_{i+1}, [r]_{k-2} \rightarrow \varepsilon \mid r \in \text{lab}(P), 0 \leq i \leq k-3 \} \\ \cup \{ A \rightarrow F, X \rightarrow \varepsilon \mid A \in N \}.$$

Here any label symbol of the form  $[r]_0$ ,  $r \in \text{lab}(P)$ , must be rewritten since it belongs to the domain of  $P_{\text{ter}}$ .

For the  $\text{fw}$  restriction we need two components instead of  $P_{\text{ter}}$ , defined as

$$P'_{\text{ter},1} = \{ X \rightarrow X', X' \rightarrow X' \} \cup \{ A \rightarrow F \mid A \in N \}$$

and

$$P'_{\text{ter},2} = \{ [r]_i \rightarrow [r]_{i+1}, [r]_{k-2} \rightarrow \varepsilon \mid r \in \text{lab}(P), 0 \leq i \leq k-3 \} \\ \cup \{ X' \rightarrow \varepsilon \}.$$

Here we make use of the derivation mode which guarantees that at least one symbol is rewritten in the weak leftmost manner.

The constructions above work in both the  $=k$  and  $\geq k$  modes for any  $k \geq 2$ , thus we have shown that systems using the sw, ww or fw restrictions working in these derivation modes can simulate leftmost derivations of context-free programmed grammars with appearance checking, hence can generate any recursively enumerable language.  $\square$

A similar construction can be used to simulate context-free programmed grammars without appearance checking in the fs manner of leftmostness.

**Theorem 4.8 ([23, 25])** For  $k \geq 2$ ,

$$\mathcal{L}(P, \text{CF}) \subseteq \mathcal{L}_{\text{fs}}(\text{CD}_{\infty}, =k) \text{ and } \mathcal{L}(P, \text{CF}) \subseteq \mathcal{L}_{\text{fs}}(\text{CD}_{\infty}, \geq k).$$

**Proof.** The idea of the proof is the same as in the proof of Theorem 4.7: For any  $L \in \mathcal{L}(P, \text{CF})$ , we simulate the programmed grammar  $G = (N, T, S, P)$  without appearance checking and with  $L = L_{\text{left}}(G)$  which, according to the proof of Lemma 1.4.8 of [41], always exists.

Consider the CD grammar system  $\Gamma$  with  $P_0, P_{\text{succ}}, P'_{r,\text{succ}}$  from the proof of Theorem 4.7, for each  $(r : A \rightarrow \alpha, \sigma(r)) \in P$ . Now, the rules  $A \rightarrow B\alpha$  and  $B \rightarrow \varepsilon$  have to be applied in the first, free derivation step of the components  $P'_{r,\text{succ}}$  and  $P_{\text{succ}}$ , respectively. Furthermore, we add the component

$$P'_{\text{ter}} = \{ [r]_i \rightarrow [r]_{i+1}, [r]_{k-3} \rightarrow \varepsilon \mid r \in \text{lab}(P), 0 \leq i \leq k-4 \} \cup \\ \{ A \rightarrow F, X \rightarrow Y, Y \rightarrow \varepsilon \mid A \in N \}$$

where  $Y$  is a new nonterminal. The rule  $X \rightarrow Y$  is needed in order to prevent the cancellation of  $X$  in the first, free derivation step of  $P'_{\text{ter}}$ , hence to guarantee that no symbol  $A \in N$  is left.

This system generates the language  $L$  in the  $=k$  or  $\geq k$  modes of derivation and fs manner of leftmostness.  $\square$

We continue with the investigation of the wf mode.

**Theorem 4.9 ([23, 25])**

1.  $\mathcal{L}_{\text{wf}}(\text{CD}_{\infty}, =2) = \mathcal{L}(\text{RE})$ ,
2.  $\mathcal{L}(\text{rP}, \text{CF}, \text{ac}) \subseteq \mathcal{L}_{\text{wf}}(\text{CD}_{\infty}, =k)$  for any  $k \geq 3$ , and
3.  $\mathcal{L}(\text{rP}, \text{CF}, \text{ac}) \subseteq \mathcal{L}_{\text{wf}}(\text{CD}_{\infty}, \geq k)$  for any  $k \geq 2$ .

**Proof.** To prove the first statement, the same construction is sufficient as in the proof of Theorem 4.7 for the fw restriction and  $k = 2$ .

Next, let  $G$  be an arbitrary context-free recurrent programmed grammar. To prove the second and the third statement, we take the construction of Theorem 4.7 for the fw restriction, now simulating a free derivation of the given grammar  $G$ , but instead of  $P'_{r,\text{succ}}$ , we have

$$\begin{aligned} P''_{r,\text{succ}} = & \{ [r]_i \rightarrow [r]_{i+1} \mid 0 \leq i \leq k-3 \} \cup \{ [r]_{k-2} \rightarrow [r]'_0, A \rightarrow \alpha \} \\ & \cup \{ [p]_i \rightarrow F, [p]'_i \rightarrow F, (q)_i \rightarrow F \mid 0 \leq i \leq k, p, q \in \text{lab}(P), \\ & p \neq r \}, \end{aligned}$$

and the production  $B \rightarrow \varepsilon$  is cancelled from the set  $P_{\text{succ}}$ .

To see that the context-free recurrent programmed grammar  $G$  is simulated, note the following fact. The only difference between the behaviour of the systems working in the  $=2$  mode on the one hand, and in the  $=k$  and  $\geq k$  modes, for some  $k > 2$ , on the other hand, is that in the latter case it is possible to repeat a successful rule application more than once while completing a “label changing cycle” from  $[r]_0$  to  $[r]_{k-2}$  and then to  $[s]_0$ . Now, since the rules  $(r : A \rightarrow \alpha, \sigma(r), \phi(r))$  of a recurrent programmed grammar have the property that  $r$  is always contained in  $\sigma(r)$ , this causes no problems during the simulation of a recurrent programmed grammar. This is also the reason why the symbol  $B$  of  $P_{r,\text{succ}}$  is not needed anymore. Thus, the statements are proved.  $\square$

#### 4.2.4 The Full Competence Mode of Derivation

Let us recall that in the full (competence) mode of derivation a component is applicable to a sentential form if and only if it contains rules for each nonterminal occurring in the sentential form. This means that a leftmost derivation step is exactly the same according to the s or w restrictions, thus we have the following statement.

**Observation 4.10** ([23, 25]) For any  $n \geq 1$ ,

1.  $\mathcal{L}_{\text{sf}}(\text{CD}_n, \text{full}) = \mathcal{L}_{\text{wf}}(\text{CD}_n, \text{full})$ ,
2.  $\mathcal{L}_{\text{fs}}(\text{CD}_n, \text{full}) = \mathcal{L}_{\text{fw}}(\text{CD}_n, \text{full})$ ,
3.  $\mathcal{L}_{\alpha}(\text{CD}_n, \text{full}) = \mathcal{L}_{\alpha'}(\text{CD}_n, \text{full})$ , for  $\alpha, \alpha' \in \{\text{ss}, \text{sw}, \text{ws}, \text{ww}\}$ .

In the next theorem, we treat the leftmost restrictions appearing in the first and the second item.

**Theorem 4.11** ([23, 25]) For  $\alpha \in \{\text{sf}, \text{wf}, \text{fs}, \text{fw}\}$ ,  $\mathcal{L}_{\alpha}(\text{CD}_{\infty}, \text{full}) = \mathcal{L}(\text{RE})$ .

**Proof.** First, let  $\alpha \in \{\text{sf}, \text{wf}\}$ . As  $\mathcal{L}(\text{CD}_{\infty}, \text{full}) = \mathcal{L}(\text{RE})$ , it is sufficient to show that  $\mathcal{L}(\text{CD}_{\infty}, \text{full}) \subseteq \mathcal{L}_{\alpha}(\text{CD}_{\infty}, \text{full})$ . (The converse inclusion can be proved by construction of a Turing machine.) Given an arbitrary CD grammar system  $\Gamma = (N, T, S, P_1, P_2, \dots, P_n)$  with context-free components, we construct the system  $\Gamma' = (N \cup \{S', X\}, T, S', P_0, P'_1, P'_2, \dots, P'_n)$  with

$$P_0 = \{S' \rightarrow XS\},$$

and

$$P'_i = P_i \cup \{X \rightarrow X, X \rightarrow \varepsilon\} \text{ for } 1 \leq i \leq n.$$

We show that  $L(\Gamma, \text{full}) = L_{\alpha}(\Gamma', \text{full})$ . The derivation must be started with  $P_0$ , and then each sentential form  $Xw$  of  $\Gamma'$  corresponds to a sentential form  $w$  of  $\Gamma$ . Since the rule  $X \rightarrow X$

is added to each component, any component of  $\Gamma'$  is competent on  $Xw$  if and only if the corresponding component of  $\Gamma$  is competent on  $w$ . And since  $X$  is the leftmost nonterminal, as long as it is present, derivations with the sf and wf restrictions give  $Xw'$  as a result, if and only if  $w'$  can be produced by  $\Gamma$ .

Keeping  $X$  in the sentential form as long as necessary, any free derivation of  $\Gamma$  can be simulated by  $\Gamma'$ . If  $X$  is erased before a terminal word is produced, then the derivations of  $\Gamma'$  correspond to special full mode derivations of  $\Gamma$ , but no word which is not in  $L(\Gamma, \text{full})$  can be produced, thus,  $L(\Gamma, \text{full}) = L_\alpha(\Gamma', \text{full})$  which proves our statement for  $\alpha \in \{\text{sf}, \text{wf}\}$ .

To prove the theorem for  $\alpha \in \{\text{fs}, \text{fw}\}$ , let  $L \in \mathcal{L}(\text{RE})$  be an arbitrary recursively enumerable language, and let us consider a context-free programmed grammar  $G = (N, T, S, P)$  with appearance checking, such that  $L = L(G)$ . Let the rules of  $P$  be labelled by labels from the set  $\text{lab}(P)$ , and let them be denoted as  $(r : A \rightarrow \alpha, \sigma(r), \phi(r))$  where  $\sigma(r), \phi(r) \subseteq \text{lab}(P)$ ,  $r \in \text{lab}(P)$  are the success and failure fields of the rule  $r$ . Let us further assume, without the loss of generality, that  $r \notin \sigma(r)$ . We construct a CD grammar system  $\Gamma''$  with  $L_\alpha(\Gamma'', \text{full}) = L(G)$  having a nonterminal alphabet  $N'$ , terminal alphabet  $T$ , and start symbol  $S' \in N'$  where

$$N' = N \cup \{S'\} \cup \{A_r, [r] \mid r \in \text{lab}(P)\},$$

and the components are defined as follows.

$$P_0 = \{S' \rightarrow [r]S \mid r \in \text{lab}(P)\}$$

is the component initiating the derivation process which is continued by simulating the rule labelled with  $r \in \text{lab}(P)$ . The rules are simulated in several rewriting steps.

To simulate the successful application of the rules, we have two components for each  $(r : A \rightarrow \alpha, \sigma(r), \phi(r))$  defined as

$$P_{r,\text{succ}} = \{A \rightarrow A_r, [r] \rightarrow [r], B \rightarrow B \mid B \in N\},$$

$$P'_{r,\text{succ}} = \{A_r \rightarrow \alpha, [r] \rightarrow [s], B \rightarrow B \mid s \in \sigma(r), B \in N\},$$

where  $A$  and  $A_r$  are to be rewritten in the first, free step of  $P_{r,\text{succ}}$  and  $P'_{r,\text{succ}}$ , respectively. If  $[r] \rightarrow [s]$  is applied in the first step, then the symbols  $[s] \neq [r]$  and  $A_r$  are present in the sentential form, and the derivation according to the full mode is blocked. To simulate the “unsuccessful” application of the rules, we have for each  $r$  as above

$$P_{r,\text{fail}} = \{[r] \rightarrow [s], B \rightarrow B \mid s \in \phi(r), B \in N \setminus \{A\}\}.$$

To finish the derivation the marker has to be erased. This can be done by the component

$$P_{\text{ter}} = \{[r] \rightarrow \varepsilon \mid r \in \text{lab}(P)\}.$$

The correct use of the components is guaranteed by the full competence mode of derivation. In particular,  $P_{\text{ter}}$  can be applied only to sentential forms in  $[r]T^*$ .  $\square$

In contrast to the theorem above, we can show that CD grammar systems in the full mode of derivation with the other leftmost restrictions generate only context-free languages. More precisely, we find the following equality.

**Theorem 4.12 ([23, 25])** For  $\alpha \in \{\text{ss}, \text{sw}, \text{ws}, \text{ww}\}$ ,  $\mathcal{L}_\alpha(\text{CD}_\infty, \text{full}) = \mathcal{L}(\text{CF})$ .

**Proof.** Let  $\Gamma = (N, T, S, P_1, P_2, \dots, P_n)$  be a context-free CD grammar system. We construct a random context grammar  $G = (N', T, S, P)$  such that  $L_\alpha(\Gamma, \text{full}) = L_{\text{left}}(G)$ , where  $L_{\text{left}}(G)$  denotes the language generated by  $G$  in such a way that in each derivation step, always the leftmost nonterminal is rewritten. This is sufficient to prove our statement, since according to Lemma 1.4.4 of [41], random context grammars with this type of leftmost rewriting characterize the family of context-free languages.

Let

$$N' = N \cup \{ [A]_i, [A]'_i \mid A \in N, 1 \leq i \leq n \},$$

and let the random context rules of  $G$  be constructed as follows. To start the derivation, we need for each  $i$  with  $S \in \text{dom}(P_i)$  a rule

$$(S \rightarrow [S]_i, \emptyset, \emptyset).$$

For each  $i$ ,  $1 \leq i \leq n$ , and rule  $A \rightarrow \alpha \in P_i$  with  $(\text{symb}(\alpha) \cap N) \subseteq \text{dom}(P_i)$ , we have

$$([A]_i \rightarrow \text{col}_i(\alpha), \emptyset, N' \setminus \{ [A]_i \mid A \in \text{dom}(P_i) \}),$$

where, for  $\alpha = x_1 x_2 \dots x_t$ , the notation  $\text{col}_i(\alpha)$  is used to designate the string

$$\text{col}_i(x_1) \text{col}_i(x_2) \dots \text{col}_i(x_t)$$

with  $\text{col}_i(A) = [A]_i$  for  $A \in N$ , and  $\text{col}_i(x) = x$  for  $x \in T$ .

For each rule  $A \rightarrow \alpha \in P_i$  where  $\alpha = \alpha_0 B_1 \alpha_1 B_2 \dots B_t \alpha_t$  with  $(\text{symb}(\alpha_k) \cap N) \subseteq \text{dom}(P_i)$ ,  $0 \leq k \leq t$ , and  $B_l \in N \setminus \text{dom}(P_i)$ ,  $B_l \in \text{dom}(P_j)$ ,  $1 \leq l \leq t$ , we have

$$(A_i \rightarrow \text{col}_i(\alpha_0) [B_1]'_j \text{col}_i(\alpha_1) [B_2]'_j \dots [B_t]'_j \text{col}_i(\alpha_t), \emptyset, N' \setminus \{ [A]_i \mid A \in \text{dom}(P_i) \}).$$

Note that if there is no  $j$ , such that  $B_l \in \text{dom}(P_j)$  for all  $1 \leq l \leq t$ , then the rule cannot be used in any derivation, since its application blocks the work of  $\Gamma$ . In this case, there is no need to simulate it in  $G$ .

Let us further add, for each  $j$ ,  $1 \leq j \leq n$ ,  $A \in N$ , and  $C \in \text{dom}(P_j)$ , all rules of the forms

$$([A]_i \rightarrow [A]'_j, \{ [C]'_j \}, N' \setminus (\{ [A]_i \mid A \in \text{dom}(P_i) \} \cup \{ [A]'_j \mid A \in N \})),$$

and

$$([C]'_j \rightarrow [C]_j, \emptyset, \{ [B]_i \mid B \in N, i \neq j \})$$

to  $P$ .

By this construction, for any of the leftmost restrictions  $\alpha \in \{ \text{ss}, \text{sw}, \text{ws}, \text{ww} \}$ .  $\square$

### 4.3 Conclusion

The main results of the chapter are summarized in Table 4.1 ([23, 25]). In this table, let  $k \geq 2$ . A language family  $\mathcal{L}$  at the intersection of the row marked by  $\alpha\beta$  and the column marked by  $\gamma$  means that  $\mathcal{L}_{\alpha\beta}(\text{CD}_\infty, \gamma) = \mathcal{L}$ . The appearance of  $\mathcal{L} \subseteq \cdot$  or  $\mathcal{L} \subset \cdot$  means that  $\mathcal{L} \subseteq \mathcal{L}_{\alpha\beta}(\text{CD}_\infty, \gamma)$  or  $\mathcal{L} \subset \mathcal{L}_{\alpha\beta}(\text{CD}_\infty, \gamma)$ , respectively.

|      | trivial modes | $t$ -mode                   | $=k$ -mode                                           | $\geq k$ -mode                               | full-mode |
|------|---------------|-----------------------------|------------------------------------------------------|----------------------------------------------|-----------|
| free | CF            | ETOL                        | $\text{CF} \subset \cdot \subseteq \text{P}$         | $\text{CF} \subset \cdot \subseteq \text{P}$ | RE        |
| ss   | CF            | CF                          | CF                                                   | CF                                           | CF        |
| sw   | CF            | $\text{ETOL} \subset \cdot$ | RE                                                   | RE                                           | CF        |
| sf   | CF            | $\text{ETOL} \subset \cdot$ | $\text{CF} \subset \cdot$                            | $\text{CF} \subset \cdot$                    | RE        |
| ws   | CF            | $\text{CF} \subset \cdot$   | CF                                                   | CF                                           | CF        |
| ww   | CF            | ETOL                        | RE                                                   | RE                                           | CF        |
| wf   | CF            | ETOL                        | $\text{rP}_{\text{ac}} \subseteq \cdot$ <sup>1</sup> | $\text{rP}_{\text{ac}} \subseteq \cdot$      | RE        |
| fs   | CF            | $\text{CF} \subset \cdot$   | $\text{P} \subseteq \cdot$                           | $\text{P} \subseteq \cdot$                   | RE        |
| fw   | CF            | ETOL                        | RE                                                   | RE                                           | RE        |

Table 4.1: Power of leftmost restricted CD grammar systems summarized, compared with non-restricted CD grammar systems (row “free”).

Besides the results summarized in the table, we also presented some findings on the power of those CD grammar systems with a restricted number of components. Concerning the  $=k$  mode of derivation, we proved the inclusions  $\mathcal{L}_{\text{ww}}(\text{CD}_{\infty, =k}) \subseteq \mathcal{L}_{\alpha}(\text{CD}_{\infty, =k+1})$ , for  $\alpha \in \{\text{sw}, \text{ww}\}$ . Thus, we settled a problem for some types of leftmost rewriting which is longstanding open in the general case.

---

<sup>1</sup>In the  $= 2$ -mode, we even proved equality to RE.

## Chapter 5

# Deterministic Parsing for CD Grammar Systems

The notion of context-free  $LL(k)$  grammars is extended to CD grammar systems using two different derivation modes. The properties of the resulting language families and the possibility of parsing these languages deterministically, without backtracking, are examined.

A first approach towards the deterministic parsing of languages generated by CD grammar systems has been taken by Mitrana and Mihalache [77], with a continuation in [80]. Mainly, the generative power of CD grammar systems is investigated that obey syntactical constraints similar to those of strict deterministic context-free grammars. In the latter paper, also accepting CD grammar systems are taken into consideration, which are restricted such that every accepting derivation has a unique leftmost accepting derivation. In this thesis, we strictly follow the concepts known from deterministic top-down parsing of context-free languages as presented in Section 1.2.3. In the preceding chapter, CD grammar systems with leftmost derivations have been investigated. For the purpose of deterministic top-down parsing, the ss- and sw-types of leftmostness will be used.<sup>1</sup> Though CD grammar systems gain more generative power under the sw-leftmost restriction, a decrease of the nondeterminism during the derivation is effected: in any step of the derivation, it is determined which occurrence of a nonterminal symbol has to be replaced next. In order to allow deterministic top-down parsing, CD grammar systems are to be further restricted to unambiguity. For this, an appropriate  $LL(k)$  condition is imposed on the systems. That is, given a terminal word  $w$  (to be syntactically analyzed), for any sentential form of a leftmost derivation, the first  $k$  symbols of  $w$  which have not yet been derived to the left of the leftmost nonterminal in the sentential form determine the next step to be performed by the CD grammar system. This means, for any word of the language generated in some derivation mode, there is a unique sequence of components to be activated and, for each of these components, there is a unique sequence of productions to be applied.

First, the concept of  $LL(k)$  context-free CD grammar systems working in the  $=m$ - and  $t$ -modes of derivation,  $m \geq 1$ , using the ss- and sw-types of leftmost restrictions is defined. We prove some first hierarchical properties for the families of languages generated by these systems. Then we focus on the  $=m$ -mode of derivation. We show that  $LL(k)$  CD grammar systems working in the  $=m$ -mode, for any  $m \geq 2$ , can be simulated by  $LL(k)$  CD grammar

---

<sup>1</sup>Note that these types of derivations were also considered in [39] where they were called “strong-leftmost” and “weak-leftmost”, respectively.



systems working in the =2-mode if the so-called sw-type of leftmostness is imposed, and that  $LL(k)$  systems of this type can generate non-semilinear languages. Next we define the notion of a (strong) lookup table which, based on pairs of nonterminals and lookahead strings, identifies the component and the sequence of rules needed for the continuation of the derivation according to the  $LL(k)$  condition. Opposed to the case of  $LL(k)$  context-free grammars, the existence and the effective constructibility of the lookup table is not obvious, but we show that in most cases, if we have the lookup table, then a lookup string of length one is sufficient, and a parsing algorithm of strictly sub-quadratic time complexity can be given. Finally, we present a decidable condition which implies the effective constructibility of the lookup table.

## 5.1 Definitions

Let  $G = (V_N, V_T, S, P_1, P_2, \dots, P_n)$  be a CD grammar system with  $n$  context-free components. We call  $G$  *deterministic* ([24]), if for all  $1 \leq i \leq n$ ,  $P_i$  contains at most one rule for each nonterminal, that is, for each symbol  $A \in V_N$ , the property  $\#\{w \mid A \rightarrow w \in P_i\} \leq 1$  holds.

For  $x, y \in (V_N \cup V_T)^*$ , let  $\text{prod}(x \xrightarrow[\alpha]{\mu} y)$  denote the set of production sequences which can be used in the derivation step  $x \xrightarrow[\alpha]{\mu} y$ ,  $\mu \in \{t, =m \mid m \geq 1\}$ ,  $\alpha \in \{ss, sw\}$ . More precisely,

$$(A_0 \rightarrow w_0, A_1 \rightarrow w_1, \dots, A_{l-1} \rightarrow w_{l-1}) \in \text{prod}(x \xrightarrow[\alpha]{\mu} y)$$

if and only if there are strings  $x_0, x_1, \dots, x_l$ ,  $l \geq 1$ , such that the derivation

$$x_0 \Rightarrow x_1 \Rightarrow \dots \Rightarrow x_l$$

is of the form  $x \xrightarrow[\alpha]{\mu} y$  and  $x_{j-1} = z_{j-1}A_{j-1}z'_{j-1}$  and  $x_j = z_{j-1}w_{j-1}z'_{j-1}$ , for some  $z_{j-1}, z'_{j-1}$  and  $A_{j-1} \rightarrow w_{j-1} \in P_i$ ,  $1 \leq j \leq l$ .

For  $\alpha \in \{ss, sw\}$ ,  $\mu \in \{t, =m \mid m \geq 1\}$ , and  $x, y \in (V_N \cup V_T)^*$ , the relation  $x \xrightarrow[\alpha]{\mu} y$ , expresses the fact that there is an  $\alpha$ -leftmost derivation by  $G$  consisting of an arbitrary number of  $\mu$ -mode derivation steps yielding  $y$  from  $x$ , that is, either  $x = y$ , or

$$x \xrightarrow[\alpha]{\mu} x_1 \xrightarrow[\alpha]{\mu} x_2 \dots \xrightarrow[\alpha]{\mu} x_r = y,$$

for some  $i_j$  with  $1 \leq i_j \leq n$ ,  $1 \leq j \leq r$ .

Next, we introduce the  $LL(k)$  condition appropriate for (deterministic) CD grammar systems. It is adopted from the context-free case.

**Definition 5.1** Let  $G = (V_N, V_T, S, P_1, P_2, \dots, P_n)$  be a CD grammar system with  $n$  context-free components,  $n \geq 1$ ,  $\mu \in \{t, =m \mid m \geq 1\}$  and  $\alpha \in \{ss, sw\}$ . The system  $G$  satisfies the  $LL(k)$  condition with respect to  $\mu$  and  $\alpha$  for some  $k \geq 1$ , if for any two leftmost derivations of type  $\alpha \in \{ss, sw\}$  and of mode  $\mu \in \{t, =m \mid m \geq 1\}$ ,

$$S \xrightarrow[\alpha]{\mu} uXy \xrightarrow[\alpha]{\mu} uz \xrightarrow[\alpha]{\mu} uv \text{ and } S \xrightarrow[\alpha]{\mu} uXy \xrightarrow[\alpha]{\mu} uz' \xrightarrow[\alpha]{\mu} uv'$$

with  $u, v, v' \in V_T^*$ ,  $X \in V_N$ ,  $y, z, z' \in (V_N \cup V_T)^*$ , if  $\text{pref}_k(v) = \text{pref}_k(v')$ , then  $i = i'$  and  $\text{prod}(uXy \xrightarrow[\alpha]{\mu} uz) = \text{prod}(uXy \xrightarrow[\alpha]{\mu} uz')$  is a singleton set.

The idea behind this concept is the following ([24]): Given a terminal word  $uv$  and a sentential form  $uXy$ ,  $X \in V_N$  and  $y \in (V_N \cup V_T)^*$ , which has been obtained from  $S$ , then the first  $k$

letters of  $v$  (if they exist) allow to determine the component and the sequence of rules of that component which is to be applied to  $uXy$  in order to derive  $uv$ .

Note that, also according to the sw-type of leftmostness,  $X$  must be the very leftmost occurrence of a nonterminal in the sentential form since in those situations a new component has to start over working on the sentential forms.

In the notation of the language families  $\mathcal{L}_\alpha(\text{CD}_n, \gamma)$  for some type of leftmost restriction  $\alpha$ , mode of derivation  $\gamma$  and  $n \in \mathbb{N} \cup \{\infty\}$ , CD is replaced with dCD when only deterministic CD grammar systems are considered. If we restrict to grammar systems of degree  $n$  which satisfy the LL( $k$ )-condition for some  $k \geq 1$ , then the families of languages obtained are denoted by  $\text{CD}_n\text{LL}(k)(\mu, \alpha)$  and  $\text{dCD}_n\text{LL}(k)(\mu, \alpha)$ .

Some of the proofs from Chapter 4 and [39] can easily be adapted to the deterministic case. Thus, the hierarchy results about CD grammar systems can be supplemented as follows ([24]):

1.  $\mathcal{L}_{\text{ss}}(\text{CD}_\infty, \mu) = \mathcal{L}_{\text{ss}}(\text{dCD}_\infty, \mu) = \mathcal{L}(\text{CF})$ , for  $\mu \in \{t, =m \mid m \geq 1\}$ ,
2.  $\mathcal{L}(\text{ET0L}) \subset \mathcal{L}_{\text{sw}}(\text{CD}_3, t) = \mathcal{L}_{\text{sw}}(\text{CD}_\infty, t)$ , and
3.  $\mathcal{L}_{\text{sw}}(\text{CD}_\infty, =m) = \mathcal{L}_{\text{sw}}(\text{dCD}_\infty, =m) = \mathcal{L}(\text{RE})$ , for any  $m \geq 2$ .

Before continuing, we illustrate the notion of LL( $k$ ) CD grammar systems with two examples.

**Example 5.1** ([24]) Consider the deterministic CD grammar system

$$G_1 = (\{S, S', S'', A, B, C, A', B', C'\}, \{a, b, c\}, S, P_1, P_2, P_3)$$

with

$$\begin{aligned} P_1 &= \{S \rightarrow S', S' \rightarrow S'', S'' \rightarrow ABC, A' \rightarrow A, B' \rightarrow B, C' \rightarrow C\}, \\ P_2 &= \{A \rightarrow aA', B \rightarrow bB', C \rightarrow cC'\}, \\ P_3 &= \{A \rightarrow a, B \rightarrow b, C \rightarrow c\}. \end{aligned}$$

This system generates by sw-leftmost derivations in the (=3) mode the language

$$L_1 = \{a^n b^n c^n \mid n \geq 1\},$$

and satisfies the LL(2) condition, what is seen as follows. Consider a derivation

$$S \xrightarrow[\text{sw}]{=3}^* uXy \xrightarrow[\text{sw}]{=3}_i uz \xrightarrow[\text{sw}]{=3}^* uv;$$

the pair  $X \in V_N$  and  $\text{pref}_2(v)$  determines the component  $P_i$  and the unique production sequence in  $\text{prod}(uXy \xrightarrow[\text{sw}]{=3}_i uz)$  which are used, as indicated in the following table.<sup>2</sup>

|      | $aa$                                                                      | $ab$                                                                      |
|------|---------------------------------------------------------------------------|---------------------------------------------------------------------------|
| $S$  | $P_1:$<br>( $S \rightarrow S', S' \rightarrow S'', S'' \rightarrow ABC$ ) | $P_1:$<br>( $S \rightarrow S', S' \rightarrow S'', S'' \rightarrow ABC$ ) |
| $A$  | $P_2:$<br>( $A \rightarrow aA', B \rightarrow bB', C \rightarrow cC'$ )   | $P_3:$<br>( $A \rightarrow a, B \rightarrow b, C \rightarrow c$ )         |
| $A'$ | $P_1:$<br>( $A' \rightarrow A, B' \rightarrow B, C' \rightarrow C$ )      | $P_1:$<br>( $A' \rightarrow A, B' \rightarrow B, C' \rightarrow C$ )      |

<sup>2</sup>Such table is called *lookup table* for CD grammar systems which is considered in more detail in Section 5.3.

Thus,  $L_1 = \{a^n b^n c^n \mid n \geq 1\} \in \text{dCD}_3\text{LL}(2)(=3, \text{sw})$ . It is an easy exercise to prove that  $L_1 \in \text{dCD}_3\text{LL}(2)(=2, \text{sw})$  also holds. (Let  $A$  produce an  $a$  and  $b$  simultaneously.)

The languages  $L_2 = \{wcw \mid w \in \{a, b\}^*\}$  and  $L_3 = \{a^n b^m c^n d^m \mid n, m \geq 1\}$  are shown to be in  $\text{dCD}_5\text{LL}(2)(=2, \text{sw})$  and in  $\text{dCD}_3\text{LL}(2)(=2, \text{sw})$ , respectively, using the systems

$$G_2 = (\{S, S', A, B, A', B'\}, \{a, b, c\}, S, P_1, P_2, \dots, P_5)$$

with

$$\begin{aligned} P_1 &= \{S \rightarrow S', S' \rightarrow c, A \rightarrow b, B \rightarrow b\}, \\ P_2 &= \{S \rightarrow S', S' \rightarrow AcB, A' \rightarrow A, B' \rightarrow B\}, \\ P_3 &= \{A \rightarrow aA', B \rightarrow aB'\}, \\ P_4 &= \{A \rightarrow bA', B \rightarrow bB'\}, \\ P_5 &= \{A \rightarrow a, B \rightarrow a\}, \end{aligned}$$

and

$$G_3 = (\{S, S', A, B, A', B'\}, \{a, b, c, d\}, S, P_1, P_2, P_3)$$

with

$$\begin{aligned} P_1 &= \{S \rightarrow S', S' \rightarrow AC, A' \rightarrow A, B' \rightarrow B, C' \rightarrow C, D' \rightarrow D\}, \\ P_2 &= \{A \rightarrow aA', C \rightarrow cC', B \rightarrow b, D \rightarrow d\}, \\ P_3 &= \{A \rightarrow aB', C \rightarrow cD', B \rightarrow bB', D \rightarrow dD'\}. \end{aligned}$$

Furthermore, the languages of the three CD grammar systems do not change if the t-mode of derivation and the sw-type of leftmostness are used; then  $G_1$ ,  $G_2$  and  $G_3$  still satisfy the LL(2) condition. Hence, the languages  $L_1$ ,  $L_2$ , and  $L_3$  from the definition of mild context-sensitivity are contained in both  $\text{CD}_\infty\text{LL}(2)(=2)$  and  $\text{CD}_\infty\text{LL}(2)(\text{t})$ .

The CD grammar system of the next example is working in the ss-type of leftmostness and the t-mode of derivation. Although these systems generate only context-free languages, the LL( $k$ ) variants are also able to deterministically describe languages (and thus allow their deterministic top-down parsing) which cannot be generated by context-free LL( $k$ ) grammars.

**Example 5.2 ([24])** Consider  $G = (\{S, S', A\}, \{a, b, c\}, S, P_1, P_2, P_3, P_4, P_5)$  with components

$$\begin{aligned} P_1 &= \{S \rightarrow aS'A\}, & P_2 &= \{S' \rightarrow S\}, & P_3 &= \{S' \rightarrow \lambda\}, \\ P_4 &= \{A \rightarrow b\}, & P_5 &= \{A \rightarrow c\}. \end{aligned}$$

This deterministic system generates by leftmost derivations of type ss, in the t-mode the non-LL( $k$ ) language  $L_{\text{ss}}(G, \text{t}) = \{a^n b^n \mid n \geq 1\} \cup \{a^n c^n \mid n \geq 1\}$ , and as can easily be checked, satisfies the LL(1) condition.

## 5.2 On the Power of LL( $k$ ) CD Grammar Systems

First we present the following trivial hierarchies.

**Lemma 5.3 ([24])** For any integers  $k \geq 1$ ,  $n \geq 1$ ,  $\mu \in \{\text{t}, =m \mid m \geq 1\}$ ,  $\alpha \in \{\text{ss}, \text{sw}\}$ , and  $X \in \{\text{CD}, \text{dCD}\}$ , we have

1.  $\text{dCD}_n\text{LL}(k)(\mu, \alpha) \subseteq \text{CD}_n\text{LL}(k)(\mu, \alpha)$ ,
2.  $X_n\text{LL}(k)(\mu, \alpha) \subseteq X_n\text{LL}(k+1)(\mu, \alpha)$ ,
3.  $X_n\text{LL}(k)(\mu, \alpha) \subseteq X_{n+1}\text{LL}(k)(\mu, \alpha)$ .

Now, for any positive integer  $k$ , we summarize the relationship of the families of context-free  $\text{LL}(k)$  languages, denoted by  $\text{LL}(k)$ , and the families of languages generated by  $\text{LL}(k)$  CD grammar systems.

**Theorem 5.4 ([24])** For any  $k \geq 1$ ,

1.  $\text{LL}(k) = \text{CD}_\infty\text{LL}(k)(=1, \alpha) = \text{dCD}_\infty\text{LL}(k)(=1, \alpha)$ , for  $\alpha \in \{\text{ss}, \text{sw}\}$ ,
2.  $\text{LL}(k) \subseteq \text{dCD}_\infty\text{LL}(k)(=m, \text{ss})$ , for  $m \geq 2$ ,
3.  $\text{LL}(k) \subset \text{dCD}_\infty\text{LL}(k)(=m, \text{sw})$ , for  $m \geq 2$ ,
4.  $\text{dCD}_\infty\text{LL}(k)(t, \alpha) \setminus \text{LL}(k) \neq \emptyset$ , for  $\alpha \in \{\text{ss}, \text{sw}\}$ .

**Proof.** Let  $L \in \text{LL}(k)$  for some  $k \geq 1$ , and let  $G = (V_N, V_T, P, S)$  be a context-free  $\text{LL}(k)$  grammar with  $L = L(G)$ . Let the rules  $r \in P$  be labeled by  $1 \leq \text{lab}(r) \leq \#P$ . First we construct a deterministic CD grammar system  $G' = (V'_N, V_T, S, P_1, P_2, \dots, P_n)$  satisfying the  $\text{LL}(k)$  condition and generating  $L$  in the  $=m$ -mode of derivation, for any  $m \geq 2$ , using any of the ss or sw-types of leftmostness, as follows. The number of components of  $G'$  is going to be  $n = \#P$ . Let  $V'_N = V_N \cup \{X^{(i)} \mid 1 \leq i \leq m-1, X \in V_N\}$ , and for  $1 \leq i \leq \#P$ , if  $i = \text{lab}(X \rightarrow \alpha)$ , then let

$$P_i = \{X \rightarrow X^{(1)}, X^{(1)} \rightarrow X^{(2)}, \dots, X^{(m-1)} \rightarrow \alpha\}.$$

It is easy to see that for any  $\mu \in \{=m \mid m \geq 2\}$ ,  $\alpha \in \{\text{ss}, \text{sw}\}$ , a rewriting step  $x \xrightarrow[\alpha]{\mu}_i y$  in  $G'$  is possible if and only if  $x \Rightarrow_i y$  is possible in  $G$ , where  $\Rightarrow_i$  denotes a rewriting step on the leftmost nonterminal using rule  $r$  with  $\text{lab}(r) = i$ .

For the proof of the equality in point 1, we first note that any CD grammar system  $G = (V_N, V_T, S, P_1, P_2, \dots, P_n)$  working in the  $=1$ -mode is equivalent to a context free grammar  $(V_N, V_T, S, P)$  where  $P = \bigcup_{1 \leq i \leq n} P_i$  using any of the two types of leftmostness in the derivations. Moreover, both of them are equivalent to a deterministic CD grammar system  $G' = (V_N, V_T, S, P'_1, P'_2, \dots, P'_r)$  working in the  $=1$ -mode, where  $r = \#P$ ,  $\#P_i = 1$  for  $1 \leq i \leq r$ , and  $\bigcup_{1 \leq i \leq r} P_i = P$ . Then the equality follows from point 1 of Lemma 5.3.

The strictness of the inclusions in point 3 and the statement of 4 follows from Example 5.1 and Example 5.2 above.  $\square$

Now we study further properties of the families of languages generated by CD grammar systems satisfying the  $\text{LL}(k)$  property in the  $=m$ -mode of derivation,  $m \geq 2$ , and the sw-type of leftmostness. We show that any language in this family can also be generated by deterministic systems in the  $=2$ -mode.

**Theorem 5.5 ([24])** For any  $k \geq 1$ ,  $m \geq 2$ ,

$$\text{CD}_\infty\text{LL}(k)(=m, \text{sw}) = \text{dCD}_\infty\text{LL}(k)(=m, \text{sw}) = X_\infty\text{LL}(k)(=2, \text{sw}),$$

where  $X \in \{\text{dCD}, \text{CD}\}$ .

**Proof.** We show that the inclusion  $\text{CD}_\infty\text{LL}(k)(=m_1, \text{sw}) \subseteq \text{dCD}_\infty\text{LL}(k)(=m_2, \text{sw})$  holds for any  $m_2 \geq 2$ . Let  $G = (V_N, V_T, S, P_1, P_2, \dots, P_n)$  be a CD grammar system satisfying the LL( $k$ ) condition,  $k \geq 1$ , in derivation mode  $=m_1$  for some  $m_1 \geq 2$ , with the sw-type of leftmostness. For any  $m_2 \geq 2$ , we construct a deterministic CD grammar system  $G'$  satisfying the LL( $k$ ) condition, such that  $L_{\text{sw}}(G, =m_1) = L_{\text{sw}}(G', =m_2)$ . Let the set of terminals and the start symbol of  $G'$  be the same as that of  $G$ , let the set of nonterminals be defined as  $V'_N = V_N \cup \{X_{i,j}^{(l)} \mid 1 \leq i \leq n, 1 \leq j \leq m_1, 1 \leq l \leq m_2 - 1\}$ , the union being disjoint, and let us define for all  $P_i$ ,  $1 \leq i \leq n$ , the rule sets  $P_{i,j}$ ,  $1 \leq j \leq r_i$  for some  $r_i \geq 1$ , in such a way that for all  $1 \leq i \leq n$ ,

- $\bigcup_{1 \leq j \leq r_i} P_{i,j} = P_i$ ,
- $\text{dom}(P_{i,j}) = \text{dom}(P_i)$  for all  $1 \leq j \leq r_i$ , and
- $P_{i,j}$  is deterministic, that is, for all  $A \in \text{dom}(P_{i,j})$ ,  $\#\{w \mid A \rightarrow w \in P_{i,j}\} = 1$ .

Now for all  $i, j, l_i$ ,  $1 \leq i \leq n$ ,  $2 \leq j \leq m_1 - 1$ ,  $1 \leq l_i \leq r_i$ , let the components of  $G'$  be defined as follows.

$$\begin{aligned} P_{i,1,l_i} &= \{A \rightarrow X_{i,1}^{(1)}w \mid A \rightarrow w \in P_{i,l_i}\} \cup \\ &\quad \{X_{i,1}^{(s)} \rightarrow X_{i,1}^{(s+1)}, X_{i,1}^{(m_2-1)} \rightarrow X_{i,2}^{(1)} \mid 1 \leq s \leq m_2 - 2\}, \\ P_{i,j,l_i} &= P_{i,l_i} \cup \{X_{i,j}^{(s)} \rightarrow X_{i,j}^{(s+1)}, X_{i,j}^{(m_2-1)} \rightarrow X_{i,j+1}^{(1)} \mid 1 \leq s \leq m_2 - 2\}, \\ P_{i,m_1,l_i} &= P_{i,l_i} \cup \{X_{i,m_1}^{(s)} \rightarrow X_{i,m_1}^{(s+1)}, X_{i,m_1}^{(m_2-1)} \rightarrow \lambda \mid 1 \leq s \leq m_2 - 2\}. \end{aligned}$$

To see that the  $L_{\text{sw}}(G, =m_1) = L_{\text{sw}}(G', =m_2)$  holds, consider that for  $u \in V_T^*$ ,  $G$  can execute a derivation step  $uAy \xrightarrow[sw]{=m_1} uay'$  if and only if

$$uAy \xrightarrow[sw]{=m_2} uX_{i,1,l_i}^{(1)}y_1 \xrightarrow[sw]{=m_2} uX_{i,2,l_i}^{(1)}y_2 \dots uX_{i,m_1,l_i}^{(1)}y_{m_1-1} \xrightarrow[sw]{=m_2} uay'$$

for some  $1 \leq l_{i,j} \leq r_i$  for all  $1 \leq j \leq m_1$ , can be executed by  $G'$ . Note that the nonterminals  $X_{i,j}^{(l)}$  must always be replaced as they appear leftmost in the sentential forms. The leftmostness of these nonterminals also implies that until they have been erased, no new terminal symbols are added to the already derived terminal prefix of the generated string appearing left of these nonterminals, so the rule sequence determined by the LL( $k$ ) property at the beginning of an  $m_1$ -step derivation of  $G$  and these nonterminals of the form  $X_{i,j}^{(s)}$  also determine the unique rule sequence for each  $m_2$ -step derivation of  $G'$  which means that it also satisfies the LL( $k$ ) property.

The statements of the theorem are consequences of the inclusion we have proved above, and the results of Lemma 5.3.  $\square$

**Lemma 5.6 ([22])** For any  $X \in \{\text{dCD}, \text{CD}\}$ ,  $\mu \in \{t, =m \mid m \geq 2\}$ , there are non-semilinear languages in  $X_\infty\text{LL}(1)(\mu, \text{sw})$ .

**Proof.** (Sketch) According to Theorem 5.5 it is sufficient to present a deterministic CD grammar system which satisfies the LL(1) condition and generates a non-semilinear language in the derivation modes  $=2$  and  $t$ , with the sw-type of leftmostness. Consider the CD grammar system

$$G = (V_N, \{a, b, c, d, e, f\}, P_1, P_2, \dots, P_{13}, S)$$

with  $V_N = \{S, S', A, A', B, B', C, C', D, E, F, T, X\}$  and

$$\begin{aligned}
P_1 &= \{S \rightarrow S', S' \rightarrow AET\}, \\
P_2 &= \{A \rightarrow dA', E \rightarrow DD\}, & P_7 &= \{B \rightarrow dB', D \rightarrow EE\}, \\
P_3 &= \{A' \rightarrow X, X \rightarrow A\}, & P_8 &= \{B' \rightarrow X, X \rightarrow B\}, \\
P_4 &= \{A \rightarrow aX, X \rightarrow B\}, & P_9 &= \{B \rightarrow aX, X \rightarrow A\}, \\
P_5 &= \{A \rightarrow X, X \rightarrow C\}, & P_{10} &= \{B \rightarrow X, X \rightarrow C\}, \\
P_6 &= \{C \rightarrow eC', D \rightarrow b\}, & P_{11} &= \{C \rightarrow fC', E \rightarrow b\}, \\
P_{12} &= \{C' \rightarrow X, X \rightarrow C\}, \\
P_{13} &= \{C' \rightarrow c, D \rightarrow F, E \rightarrow F, T \rightarrow c\}.
\end{aligned}$$

We determine the language  $L_{\text{sw}}(G, =2)$ . Any leftmost derivation of  $G$  in the  $=2$ -mode starting off with the axiom must initially use component  $P_1$  leading to  $AET$ . Then, the components  $P_2, P_3, P_4, P_7, P_8,$  and  $P_9$  can be used in turns. In this phase, the number of occurrences of  $D$ 's and  $E$ 's can be increased. Note that this number can at most be doubled until a new symbol  $a$  will be introduced before a further increase is possible. Moreover, whenever one more  $D$  or  $E$  is introduced, then simultaneously a terminal  $d$  must emerge. This phase is finished by one application of either  $P_5$  or  $P_{10}$  turning the leftmost nonterminal ( $A$  or  $B$ ) to  $C$ . Now, all occurrences of nonterminals  $D$  and  $E$  can be terminated with the help of  $P_6, P_{11},$  and  $P_{12}$ . Finally, the leftmost and the rightmost nonterminals, that is  $C'$  and  $T$  at this stage of the derivation, can be terminated by using  $P_{13}$ . Since also this terminating component must be applied in the leftmost way and  $F$  is a trap symbol, it is guaranteed that all occurrences of  $D$  and  $E$  have vanished before  $P_{13}$  can successfully be applied in the  $=2$ -mode. Therefore, in every non-terminal sentential form, either  $A, B, C$  (or its primed versions or  $X$ ) is the leftmost occurring nonterminal, steering the selection of the components. Thus, the different phases of the derivation cannot be mixed.

Consequently, the non-semilinear language  $L$  is generated, where

$$\begin{aligned}
L \subseteq K &= \{fcbc\} \cup \{d^{i_1}ad^{i_2}a \dots d^{i_n}vcb^m c \mid n \geq 1, 0 \leq i_j \leq 2^j, \\
&\text{for } 1 \leq j \leq n, m = 1 + \sum_{j=1}^n i_j, v \in \{e, f\}^m\}.
\end{aligned}$$

Here,  $L$  is not equal to  $K$  only because the portion  $v$  has to obey some additional combinatorial properties which do not affect the non-semilinearity of the language. Since writing down these properties would decrease readability, they are omitted. On the other hand, the  $e$ 's and  $f$ 's are needed in order to make sure that  $G$  is  $\text{LL}(k)$  in the  $=2$ -mode. In fact, one can readily prove that  $G$  satisfies the  $\text{LL}(1)$  condition. By analyzing this system, we can see that it generates the same non-semilinear language also in the  $t$ -mode of derivation.  $\square$

### 5.3 Using Lookup Tables

The aim of this section is to provide deterministic top-down parsing methods for CD grammar systems satisfying the  $\text{LL}(k)$  property in  $\alpha$ -type leftmost derivations of mode  $\mu, \mu \in \{t, =, m\}, \alpha \in \{\text{ss}, \text{sw}\}$ . In the case of  $\text{LL}(k)$  CD grammar systems under the  $\text{ss}$ -leftmost restrictions, slight modifications of the usual top down methods for context-free  $\text{LL}(k)$  parsing can be used to provide parsing algorithms also for grammar systems of these types, as always the leftmost nonterminal is to be replaced.

For CD grammar systems working in the  $\text{sw}$ -type of leftmostness we need more sophisticated methods while derivations of this type also rewrite nonterminals which are not the

leftmost ones in the sentential form. In order to present an appropriate parsing algorithm, the notion of (*strong*) *lookup table* for CD grammar systems satisfying the  $LL(k)$  condition with respect to  $\mu$  and  $\alpha$  is needed. The lookup table determines the component and the sequence of rules which are needed for the continuation of the derivation, according to the definition of the  $LL(k)$  condition. In strong lookup tables, the selection of the rules is based on pairs of nonterminals (the leftmost nonterminal in the leftmost derivation under consideration) and lookahead strings (the first  $k$  terminal letters of the suffix of the resulting terminal word which is derived by the remaining part of the leftmost derivation).<sup>3</sup>

First, we present some notions used in the definition of the strong lookup table and the parsing algorithm, see also [22].

- A *production* is  $p = X \rightarrow \alpha \in V_N \times (V_N \cup V_T)^*$  with  $left(p) = X, right(p) = \alpha$ .
- A *stack over*  $\mathbb{N}$  is  $st = x_j]x_{j-1}] \dots ]x_1]$ ,  $x_i \in \mathbb{N}$ ,  $1 \leq i \leq j$ , with  $top(st) = x_j$ ,  $pop(st) = x_{j-1}] \dots ]x_1]$ , and for some  $y \in \mathbb{N}$ ,  $push(y, st) = y]x_j]x_{j-1}] \dots ]x_1]$ . The empty stack,  $pop(x]$  for some  $x \in \mathbb{N}$ , is denoted by  $\lambda]$ .
- A *stack over*  $V_N \cup V_T$  is  $st = x_j]x_{j-1}] \dots ]x_1]$ ,  $x_i \in V_N \cup V_T$ ,  $1 \leq i \leq j$ , with  $top(st) = x_j$ ,  $pop(st) = x_{j-1}] \dots ]x_1]$  and for some  $y = y_1 \dots y_m \in (V_N \cup V_T)^*$ ,  $y_i \in V_N \cup V_T$ ,  $1 \leq i \leq m$ ,  $push(y, st) = y_1] \dots ]y_m]x_j]x_{j-1}] \dots ]x_1]$ . The empty stack,  $pop(x]$  for some  $x \in V_N \cup V_T$ , is denoted by  $\lambda]$ .
- A *production queue* is  $pq = (p_1, p_2, \dots, p_j)$ ,  $p_i \in P_l$ ,  $1 \leq l \leq n$ ,  $1 \leq i \leq j$ , with  $first(pq) = p_1$ , but  $first(pq) = (p_2, \dots, p_j)$ .

**Definition 5.2 ([22])** Let  $G = (V_N, V_T, S, P_1, P_2, \dots, P_n)$  be a CD grammar system satisfying the  $LL(k)$  condition with respect to  $\mu$  and  $\alpha$ , for some  $\mu \in \{t, =m \mid m \geq 1\}$ ,  $\alpha \in \{ss, sw\}$ ,  $n \geq 1$ , and  $k \geq 1$ . The strong lookup table for  $G$  is given as  $lookupTable \subseteq V_N \times V_T^k \times PQ$  where  $PQ$  denotes the set of all production queues consisting of  $m$  productions; it is a function which for a nonterminal  $X \in V_N$  and a terminal word of length  $k$ ,  $y \in V_T^k$ , returns a production queue  $pq = lookupTable(X, y)$ .

In Figure 5.1 we present a parsing algorithm for languages in  $CD_nLL(k)(= m, sw)$ , see also [22, 24]. It uses the variables

- *step*, *stepOfTopmost*  $\in \mathbb{N}$ , natural numbers,
- *mainStack*, a stack over  $V_N \cup V_T$ , the “main” stack of the parser,
- *stacksForN*, an  $l$ -tuple of stacks for natural numbers where  $l = |V_N|$ ; it provides a stack over  $\mathbb{N}$  for each nonterminal of the grammar system,
- *input*  $\in V_T^*$ , the string to be analyzed,
- *topmost*  $\in V_N$ , a nonterminal symbol,
- *pQueue*, a production queue as above,
- *pQueuesLeft*  $\subseteq \mathbb{N} \times PQ$ , where  $PQ$  denotes the set of all production queues of length at most  $m$ , that is, *pQueuesLeft* is a set of pairs of the form  $(i; pq)$  where  $i$  is an integer and  $pq$  is a production queue as above,
- *pToUse*  $\in V_N \times (V_N \cup V_T)^*$ , a production as above.

```

1 step ← 0
2 mainStack ← push(mainSt, S)
3 stacksForN(S) ← push(stacksForN(S), 0)
4 while mainStack is not empty and there is no ERROR do
5 if top(mainStack) is a terminal symbol then
6 if top(mainStack) coincides with the first symbol of input then
7 mainStack ← pop(mainStack)
8 input ← input without its first symbol
9 else ERROR
10 else topmost ← top(mainStack)
11 stepOfTopmost ← top(stacksForN(topmost))
12 stacksForN(topmost) ← pop(stacksForN(topmost))
13 if there exist $(i; pQueue) \in pQueuesLeft$ such that
14 $i \geq stepOfTopmost$, $left(first(pQueue)) = topmost$,
15 and furthermore, if $(i'; pQueue') \in pQueuesLeft$
16 with $left(first(pQueue')) = topmost$, then $i < i'$, then
17 pQueuesLeft ← pQueuesLeft − $\{(i; pQueue)\}$
18 pToUse ← first(pQueue)
19 pQueue ← butfirst(pQueue)
20 if pQueue is not empty then
21 pQueuesLeft ← pQueuesLeft ∪ $\{(i; pQueue)\}$
22 mainStack ← pop(mainStack)
23 mainStack ← push(mainStack, right(pToUse))
24 for each symbol X from right(pToUse) do
25 if $X \in V_N$ then
26 stacksForN(X) ← push(stacksForN(X), step)
27 else step ← step + 1
28 lookahead ← the next k symbols of input
29 pQueue ← lookupTable(topmost, lookahead)
30 if pQueue is empty then
31 ERROR
32 else pToUse ← first(pQueue)
33 pQueue ← butfirst(pQueue)
34 if pQueue is not empty then
35 pQueuesLeft ← pQueuesLeft ∪ $\{(step, pQueue)\}$
36 mainStack ← pop(mainStack)
37 mainStack ← push(mainStack, right(pToUse))
38 for each symbol X from right(pToUse) do
39 if $X \in V_N$ then
40 stacksForN(X) ← push(stacksForN(X), step)
41 if there is no ERROR then successful termination

```

Figure 5.1: The parsing algorithm.



|       | $a$                                                                                     | $b$                                                                                      |
|-------|-----------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|
| $S$   | $P_1:$<br>( $S \rightarrow A_1A_2A_1A_3, A_2 \rightarrow b,$<br>$A_3 \rightarrow A_4$ ) | $P_1:$<br>( $S \rightarrow bA_1A_2A_1A_3, A_2 \rightarrow b,$<br>$A_3 \rightarrow A_4$ ) |
| $A_1$ | $P_2:$<br>( $A_1 \rightarrow aA_2, A_1 \rightarrow aA_2,$<br>$A_4 \rightarrow A_5$ )    | —                                                                                        |
| $A_2$ | $P_3:$<br>( $A_2 \rightarrow a, A_2 \rightarrow a, A_5 \rightarrow b$ )                 | —                                                                                        |

Figure 5.2: The lookup table for the grammar system of Example 5.7.

In the following we demonstrate the work of the algorithm through an example.

**Example 5.7 ([22])** Consider the CD grammar system

$$G = (\{S, A_1, A_2, A_3, A_4, A_5\}, \{a, b\}, S, P_1, P_2, P_3)$$

with

$$\begin{aligned} P_1 &= \{S \rightarrow A_1A_2A_1A_3, S \rightarrow bA_1A_2A_1A_3, A_2 \rightarrow b, A_3 \rightarrow A_4\}, \\ P_2 &= \{A_1 \rightarrow aA_2, A_4 \rightarrow A_5\}, \\ P_3 &= \{A_2 \rightarrow a, A_5 \rightarrow b\}, \end{aligned}$$

This system generates, in the (= 3) mode, the finite language

$$L = \{aabaab, baabaab\},$$

and satisfies the LL(1) condition. The (strong) lookup table for the parser is seen on Figure 5.2.

Let us see how the parser analyzes the string  $aabaab \in L_{\text{sw}}(G, =3)$ . This string is generated in three steps in the (= 3)-mode as follows.

$$S \xrightarrow[\text{sw}]{=3}_1 A_1bA_1A_4 \xrightarrow[\text{sw}]{=3}_2 aA_2baA_2A_5 \xrightarrow[\text{sw}]{=3}_3 aabaab.$$

Now we will follow the work of the parser step-by-step, and describe its configuration by

$$(input, mainStack, step, stacksForN, pQueuesLeft)$$

where the variables are as described above. The value of  $stacksForN$  will be denoted as  $(\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5)$  where  $\alpha_0$  is the contents of  $stacksForN(S)$ , and for  $i \in \{1, 2, 3, 4, 5\}$ ,  $\alpha_i$  is the contents of  $stacksForN(A_i)$ .

The initial configuration of the parser is

$$(aabaab, S], 0, (0], \lambda], \dots, \lambda], \emptyset),$$

<sup>3</sup>In general, lookup tables might need to exploit some further, finite piece of information about the current sentential form, see [3] for this notion for context-free grammars.

meaning that nothing is read from the input, the initial symbol,  $S$ , is placed in the main stack, the step counter is set to  $zero$ , the integer  $zero$  is placed in the stack  $stacksForN(S)$  associated to the nonterminal  $S$  which indicates that it appeared in the main stack when the counter  $step$  had value  $zero$ , and the set of production queues waiting to be applied,  $pQueuesLeft$ , is empty.

The main stack is not empty, so the parser starts the execution of the **while** loop of the algorithm at line 4. The symbol on the top of the main stack is a nonterminal, so it jumps to line 10. Since  $pQueuesLeft$ , the set of production queues waiting for execution is empty, after popping the stack  $stacksForN(S)$  associated to the symbol in the main stack, the parser proceeds with the instruction on line 24 by increasing the counter  $step$  and identifying the production queue to be applied with the help of the lookup table. At this point

$$\begin{aligned} lookahead &= a, \\ topmost &= S, \\ pQueue &= (S \rightarrow A_1A_2A_1A_3, A_2 \rightarrow b, A_3 \rightarrow A_4). \end{aligned}$$

The production to be used is the first production of  $pQueue$ ,

$$pToUse = S \rightarrow A_1A_2A_1A_3.$$

Now the remaining part of  $pQueue$  is stored in  $pQueuesLeft$  indexed with  $one$ , the current value of the  $step$  counter, as a pair  $(1; A_2 \rightarrow b, A_3 \rightarrow A_4)$ . This indicates that the rules of this queue can be used on nonterminals that appeared in the main stack when the  $step$  counter had value  $one$  or less. Now the top of the main stack is replaced with the word on the right side of  $pToUse$ , the stack associated to  $S$  is emptied, and the value of  $step$  is placed into the stacks associated to the nonterminals appearing on the right side of the rule,  $stacksForN(X)$ ,  $X \in \{A_1, A_2, A_3\}$ . The configuration of the parser is

$$(aabaab, A_1]A_2]A_1]A_3], 1, (\lambda], 1]1], 1], 1], \lambda], \lambda], \{(1; A_2 \rightarrow b, A_3 \rightarrow A_4)\}).$$

Now the parser starts the execution of the **while** loop on line 4 again. Since the top of the main stack is a nonterminal,  $A_1$ , and since there is no production queue in  $pQueuesLeft$  having  $A_1$  on the left-hand side of its first rule, after popping  $stacksForN(A_1)$ , the stack associated with the topmost nonterminal, the parser continues with line 24 of the algorithm by increasing the counter  $step$ , and determining the production queue and the production to be used with the help of the lookup table, obtaining

$$\begin{aligned} pQueue &= (A_1 \rightarrow aA_2, A_1 \rightarrow aA_2, A_4 \rightarrow A_5), \\ pToUse &= A_1 \rightarrow aA_2. \end{aligned}$$

After the application of the production  $A_1 \rightarrow aA_2$  to the topmost nonterminal of the main stack, the parser is in the configuration

$$\begin{aligned} (aabaab, a]A_2]A_2]A_1]A_3], 2, (\lambda], 1], 2]1], 1], \lambda], \lambda], \\ \{(1; A_2 \rightarrow b, A_3 \rightarrow A_4), (2; A_1 \rightarrow aA_2, A_4 \rightarrow A_5)\}), \end{aligned}$$

and then the execution of the algorithm continues at line 4 again.

Since the top of the main stack is the same terminal as the first symbol of the input, the parser enters

$$\begin{aligned} (abaab, A_2]A_2]A_1]A_3], 2, (\lambda], 1], 2]1], 1], \lambda], \lambda], \\ \{(1; A_2 \rightarrow b, A_3 \rightarrow A_4), (2; A_1 \rightarrow aA_2, A_4 \rightarrow A_5)\}) \end{aligned}$$

by popping the main stack and reading one letter of the input, then continues with line 4, and jumps to line 10 again.

Now the topmost nonterminal is  $A_2$ , and by popping *two* from the stack  $stacksForN(A_2)$ , it is clear that the production queue  $(A_2 \rightarrow b, A_3 \rightarrow A_4)$  from  $pQueuesLeft$  can not be used since it has index *one*. This means that the parser needs to turn to the lookup table again, obtaining

$$\begin{aligned} pQueue &= (A_2 \rightarrow a, A_2 \rightarrow a, A_5 \rightarrow b), \\ pToUse &= A_2 \rightarrow a. \end{aligned}$$

After the necessary replacements in the stacks and after updating the value of other variables, the parser enters

$$\begin{aligned} &(abaab, a]A_2]A_1]A_3], 3, (\lambda], 1], 1], 1], \lambda], \lambda]), \\ &\{(1; A_2 \rightarrow b, A_3 \rightarrow A_4), (2; A_1 \rightarrow aA_2, A_4 \rightarrow A_5), (3; A_2 \rightarrow a, A_5 \rightarrow b)\}. \end{aligned}$$

Now after popping the main stack and reading one more symbol of the input, the parser continues at line 10 again. This time, the topmost nonterminal is  $A_2$ , and the integer popped from the corresponding stack,  $stacksForN(A_2)$  is *one*, so the first production of the production queue  $(A_2 \rightarrow b, A_3 \rightarrow A_4)$  stored in  $pQueuesLeft$  with the same index can be used. Thus, the parser continues at line 14 of the algorithm setting

$$\begin{aligned} pToUse &= A_2 \rightarrow b, \\ pQueuesLeft &= \{(1; A_3 \rightarrow A_4), (2; A_1 \rightarrow aA_2, A_4 \rightarrow A_5), (3; A_2 \rightarrow a, A_5 \rightarrow b)\}. \end{aligned}$$

After replacing the topmost nonterminal of the main stack with the right side of  $pToUse$ , the parser enters

$$\begin{aligned} &(baab, b]A_1]A_3], 3, (\lambda], 1], \lambda], 1], \lambda], \lambda]), \\ &\{(1; A_3 \rightarrow A_4), (2; A_1 \rightarrow aA_2, A_4 \rightarrow A_5), (3; A_2 \rightarrow a, A_5 \rightarrow b)\}. \end{aligned}$$

After popping the main stack and reading one more symbol of the input, the condition on line 13 is satisfied again, so the parser sets

$$\begin{aligned} pToUse &= A_1 \rightarrow aA_2, \\ pQueuesLeft &= \{(1; A_3 \rightarrow A_4), (2; A_4 \rightarrow A_5), (3; A_2 \rightarrow a, A_5 \rightarrow b)\}, \end{aligned}$$

and then uses them, entering

$$\begin{aligned} &(aab, a]A_2]A_3], 3, (\lambda], \lambda], 3], 1], \lambda], \lambda]), \\ &\{(1; A_3 \rightarrow A_4), (2; A_4 \rightarrow A_5), (3; A_2 \rightarrow a, A_5 \rightarrow b)\}. \end{aligned}$$

Popping and reading again, then

$$\begin{aligned} pToUse &= A_2 \rightarrow a, \\ pQueuesLeft &= \{(1; A_3 \rightarrow A_4), (2; A_4 \rightarrow A_5), (3; A_5 \rightarrow b)\}, \end{aligned}$$

since the queue  $(A_2 \rightarrow a, A_5 \rightarrow b)$  stored in  $pQueuesLeft$  has index *three*, the same as the value obtained from the stack  $stacksForN(A_2)$ , so it can be used, producing

$$(ab, a]A_3], 3, (\lambda], \lambda], \lambda], 1], \lambda], \lambda]), \{(1; A_3 \rightarrow A_4), (2; A_4 \rightarrow A_5), (3; A_5 \rightarrow b)\}.$$

After the main stack is popped again and one more input symbol is read, the parser sets

$$\begin{aligned} pToUse &= A_3 \rightarrow A_4, \\ pQueuesLeft &= \{(2; A_4 \rightarrow A_5), (3; A_5 \rightarrow b)\}, \end{aligned}$$

and enters

$$(b, A_4], 3, (\lambda], \lambda], \lambda], \lambda], 1], \lambda], \{(2; A_4 \rightarrow A_5), (3; A_5 \rightarrow b)\}).$$

The value *one* is placed in the stack  $stacksForN(A_4)$  because the queue index of the rule  $A_3 \rightarrow A_4$  was *one* which means that the application of the rule happens in step *one*, that is,  $A_4$  appears in the first (= 3)-mode step of the generation of the input string.

The next configuration is

$$(b, A_5], 3, (\lambda], \lambda], \lambda], \lambda], \lambda], 2], \{(3; A_5 \rightarrow b)\},$$

and then

$$(b, b], 3, (\lambda], \lambda], \lambda], \lambda], \lambda], \lambda], \emptyset),$$

after which the last input symbol is read and the main stack is once again popped, so the parser enters

$$(\lambda, \lambda], 3, (\lambda], \lambda], \lambda], \lambda], \lambda], \lambda], \emptyset),$$

and since the main stack is empty, finishes its work at line 38 of the algorithm.

**Theorem 5.8 ([22])** If a CD grammar system  $G$  satisfying the  $LL(k)$  condition in the  $=m$  derivation mode using the sw-type of leftmostness is given together with its strong lookup table, then for  $L_{sw}(G, =m)$  a parser can be constructed as presented in Figure 5.1, which halts on every input word  $w$  in  $O(n \cdot \log^2 n)$  steps, where  $n$  is the length of  $w$ .

**Proof.** Let  $G = (V_N, V_T, S, P_1, P_2, \dots, P_s)$  be a CD grammar system satisfying the  $LL(k)$  condition in the  $(=m)$ -mode of derivation. First we show that the parser constructed according to  $G$  halts on every input.

Assume that the parser does not halt on an input word  $w \in V_T^*$ . This means that it loops infinitely, and it can only do that if the instructions on the lines 10 – 37 are executed infinitely many times. To see this, notice that the body of the main **while** loop contains one **if-then-else** statement. Instructions of the **then** part read an input symbol, so they can not be repeated infinitely many times. This implies that the **else** part on lines 10 – 37 is repeated infinitely many times.

This part of the algorithm contains an **if-then-else** statement starting with line 13, the execution of the instructions of this part mean either the execution of the **then** part on lines 14 – 23, or the **else** part on lines 24 – 37. If lines 10 – 37 are executed infinitely many times, then there must be infinitely many such executions when no terminal symbol is written on the top of the main stack in line 20 or in line 34, which means that there is an infinite sequence of consecutive executions of lines 10 – 37 during which no terminal symbol is ever written on the top of the main stack.

Since each execution of the instructions of lines 14 – 23 removes one production from the production queues stored in  $pQueuesLeft$ , the instructions on lines 24 – 37 must be executed infinitely many times, or the parser cannot loop infinitely.

Because the lookahead never changes and because the number of nonterminal symbols is finite, there must be a sequence of instructions starting with lines 24 – 37, continuing with

possibly several executions of lines 10 – 23 or lines 24 – 37, and then ending with lines 24 – 37 again, in such a way that the value of *topmost*, that is, the topmost nonterminal of the main stack, is the same at the first and at the last execution of lines 24 – 37.

Since the choice of the productions to be applied is based on the lookup table (line 26), the situation outlined above can only happen if the lookup table has certain properties which we describe below.

Let  $X \in V_N$  and  $y \in V_T^k$  be a row and a column index of the lookup table, and let  $maxchain(X, y)$  denote the production queue with the following properties:

- $maxchain(X, y)$  is a prefix  $(p_1, \dots, p_l)$  of the corresponding entry of the lookup table,  $lookupTable(X, y) = (p_1, \dots, p_l, p_{l+1}, \dots, p_m)$ .
- If  $maxchain(X, y)$  and  $lookupTable(X, y)$  are as above, then  $X \Rightarrow_{p_1} X_1 w_1$  and, furthermore,  $X_i w_i \Rightarrow_{p_{i+1}} X_{i+1} w_{i+1}$ ,  $X_i \in V_N$ ,  $w_i \in (V_N \cup V_T)^*$ , for each  $1 \leq i \leq l - 1$ , and each  $p_i$  rewrites the leftmost nonterminal, that is, it is of the form  $p_1 = X \rightarrow \alpha_1$ , and  $p_i = X_{i-1} \rightarrow \alpha_i$ ,  $2 \leq i \leq l$ , and
- $maxchain(X, y)$  contains the maximal number of productions with the properties above, that is,  $p_{l+1} = Z \rightarrow w$  where  $Z \neq X_l$ .

The parser may enter an infinite loop, if there exist a column of the lookup table, labelled with  $y \in V_T^k$ , such that

$$\begin{aligned} X &\Rightarrow_{maxchain(X,y)} X_1 w_1 \Rightarrow_{maxchain(X_1,y)} X_2 w_2 \Rightarrow_{maxchain(X_2,y)} \dots \\ \dots &\Rightarrow_{maxchain(X_l,y)} X_{l+1} w_{l+1} = X w_{l+1}, \end{aligned}$$

where  $X, X_i \in V_N$ ,  $w_i \in (V_N \cup V_T)^*$ ,  $1 \leq i \leq l + 1$ , and  $\Rightarrow_{maxchain(X,y)}$  denotes a leftmost derivation sequence using the rules of the production queue  $maxchain(X, y)$ .

Now we show that such a column cannot exist in the lookup table. If during a leftmost (=m)-mode derivation we encounter the nonterminal  $X$  as the leftmost nonterminal, and the production queue identified by  $X$  and the lookahead would be the queue in  $lookupTable(X, y)$ , then a successful application of the rules would lead to the choice of the queue  $lookupTable(X_1, y)$ ,  $lookupTable(X_2, y)$ , and so on, until we would obtain  $X$  again as the leftmost nonterminal with the same lookahead, thus, the production queues identified by the leftmost nonterminal and the lookahead would never lead to a successful derivation which is a contradiction.

Now we show that given the input word  $w \in V_T^*$ , the parser halts after  $O(n \cdot \log^2 n)$  number of steps where  $n = |w|$ . With similar arguments as above, we can show that the number of instructions executed without reading any input symbol is  $O(1)$  which means that the running time of the parser is the length of the input multiplied by the time necessary to execute an instruction. All the instructions used in the algorithm can be executed in constant time, except the evaluation of the condition on line 13 and the assignments on line 14, 18, and 32 because they require the manipulation of the data stored in the set structure *pQueuesLeft*. The evaluation of line 13 requires a search, the assignments require the addition and the deletion of an element using a set where the number of stored elements can be as many as  $O(n)$ .

All of these operations, however, can be executed in  $O(\log^2 n)$  time if we use balanced search trees, such as red-black trees for example, to store the elements of the set *pQueuesLeft*.

(For more on balanced search trees and red-black trees in particular, see [28].) The implementation of the set  $pQueuesLeft$  must consist of a red-black tree for each nonterminal  $X \in V_N$  which stores the indexed production queues  $(i; pq) \in pQueuesLeft$  with  $X = left(first(pq))$  ordered by the index  $i \in \mathbb{N}$ . Having such a structure, the evaluation of the condition on line 13 can be realized by turning to the search tree associated to the nonterminal  $topmost$  to obtain the pair  $(i; pQueue)$  where either  $i = stepOfTopmost$ , or if such index is not present, then  $i$  is the smallest available index with  $i \geq stepOfTopmost$ . To perform this search takes  $O(\log n)$  comparisons since even in the worst case when the index is not present, it is enough to explore one path of the red-black tree leading from the root to one of the leaves, and the length of these paths, that is, the height of the tree is  $O(\log n)$ . To execute lines 14, 18, and 32, that is, to add or remove elements from the structure first requires a search to determine the appropriate tree and the location of the element in the tree, and then a constant number of elements need to be manipulated to insert or to remove the data. Since the number of trees used are finite, the number of necessary comparisons and data manipulations are  $O(\log n)$ . One comparison or one data manipulating step, however, also requires  $O(\log n)$  time, since the integers used to index the production queues, that is, the keys used to index the nodes of the search tree, might be as large as  $n$ , so their representation can be as long as  $\log n$  which means that comparing, reading or writing them requires  $O(\log n)$  elementary computation steps. This gives a total running time of  $O(n \cdot \log^2 n)$  where  $n = |w|$ , the length of the input word.  $\square$

In case of deterministic systems, the lookup table is more simple, it only needs to give a component for the pairs of nonterminals and lookahead strings, the exact order of the application of the rules is automatically determined due to the restriction to sw-type of leftmost derivations.

**Definition 5.3 ([24])** Let  $G = (V_N, V_T, S, P_1, P_2, \dots, P_n)$  be a deterministic CD grammar system satisfying the  $LL(k)$  condition with respect to  $\mu$  and  $\alpha$ , for some  $\mu \in \{t, =m \mid m \geq 1\}$ ,  $\alpha \in \{ss, sw\}$ ,  $n \geq 1$ , and  $k \geq 1$ . The *strong lookup table*  $M_G$  for  $G$  is a subset of  $V_N \times V_T^{\leq k} \times \{P_1, P_2, \dots, P_n\}$  such that for all  $uXy$  with

$$S \xrightarrow[\alpha]{\mu}^* uXy \xrightarrow[\alpha]{\mu} uz \xrightarrow[\alpha]{\mu}^* uv$$

$u, v \in V_T^*$ ,  $X \in V_N$ ,  $y, z \in (V_N \cup V_T)^*$ , the entry  $(X, \text{pref}_k(v))$  contains the component which is to be applied to the sentential form  $uXy$ .

According to this definition, the lookup table for the CD grammar system from Example 5.1 is written as follows:

$$\{(S, aa, P_1), (S, ab, P_1), (A, aa, P_2), (A, ab, P_3), (A', aa, P_1), (A', ab, P_1)\}.$$

For the existence of a lookup table, it is necessary for a CD grammar system to satisfy the  $LL(k)$  condition. The implication in the other direction, opposed to the context-free case, is not as obvious. Just as unclear is the existence of a general algorithm for the construction of a lookup table when a (deterministic) grammar system is given.

However, in many cases it is not difficult to construct a lookup table, such as in the Examples presented in this chapter.

Now we show that if we assume the existence of a strong lookup table, then in the case of the sw-type of leftmostness, the length of the necessary lookahead can be decreased to  $k = 1$ .

That is, in contrast to the context-free case, the hierarchies of language families corresponding to deterministic LL( $k$ ) CD grammar systems induced by  $k$  collapse, namely to the first level.

**Theorem 5.9** ([24]) Given a deterministic CD grammar system  $G$  satisfying the LL( $k$ ) condition for some derivation mode  $\mu \in \{t, = m \mid m \geq 2\}$  with the sw-type of leftmostness, and its strong lookup table  $M_G$ , then  $L_{\text{sw}}(G, \mu) \in \text{dCD}_\infty \text{LL}(1)(\mu, \text{sw})$ .

**Proof.** Given  $G = (V_N, V_T, S, P_1, P_2, \dots, P_n)$ , a deterministic CD grammar system as above, satisfying the LL( $k$ ) condition for some  $k \geq 2$ . Let the look-up table of  $G$  denoted by  $M_G$ . We construct a context-free CD grammar system  $H$  which satisfies the LL(1) condition and for which  $L_{\text{sw}}(H, \mu) = L_{\text{sw}}(G, \mu)$  holds,  $\mu$  is as above. Let the set of non-terminals for  $H$  be the set

$$V'_N = V_N \cup \{(X, v), (X, v)^{(i)}(A, v), (A, v)', (A, v)^{(i)}, (\bar{a}, v), (\bar{a}, v)^{(i)} \mid A \in V_N, \\ v \in V_T^{\leq k}, a \in V_T, 1 \leq i < m\} \cup \{\bar{a} \mid a \in V_T\},$$

where  $X$  is a new symbol, and let the axiom of  $H$  be  $(S, \lambda)$ . Let us define for any  $\alpha = x_1 x_2 \dots x_t$ ,  $x_i \in V_N \cup V_T$ ,  $1 \leq i \leq t$ , the string  $\bar{\alpha} = b(x_1)b(x_2) \dots b(x_t)$  with  $b(x) = x$  for  $x \in V_N$ , and  $b(x) = \bar{x} \in \bar{V}_T$  for  $x \in V_T$ .

We construct the following components.

(1) *Scanning components.* For all  $A \in V_N$  and  $u \in V_T^{\leq k-1}$ ,  $v \in V_T^{\leq k}$ , we have:

$$\{(A, u) \rightarrow (A, u)^{(1)}, \dots, (A, u)^{(m-2)} \rightarrow (A, u)^{(m-1)}, (A, u)^{(m-1)} \rightarrow a(A, ua)'\},$$

and also the components:

$$\{(A, v)' \rightarrow (A, v)^{(1)}, \dots, (A, v)^{(m-2)} \rightarrow (A, v)^{(m-1)}, (A, v)^{(m-1)} \rightarrow (A, v)\}.$$

These collect the look-ahead string  $v$  of length at most  $k$  symbol by symbol. Their correct, deterministic use is guaranteed by the look-ahead of length one. If  $|u| = k$  for some  $(A, u)$ , or the look-ahead symbol is  $\lambda$ , then these components cannot be used any more.

(2) *Direct simulating components.* For  $(A, v, P_i) \in M_G$ ,  $A \rightarrow \alpha \in P_i$ ,  $\alpha = uB\beta$  with  $u \in V_T^*$ ,  $B \in V_N$ ,  $\beta \in (V_N \cup V_T)^*$ , we have:

$$\{(A, v) \rightarrow v_1(X, v_2)B\bar{\beta}\} \cup \bar{P}_i$$

where if  $u = vv'$ , for some  $v' \in V_T^*$ , then  $v_1 = v'$  and  $v_2 = \lambda$ . Otherwise, if  $v = uu'$ , for some  $u' \in V_T^*$ , then  $v_1 = \lambda$  and  $v_2 = u'$ . Furthermore,  $\bar{P}_i$  denotes the set  $\{A \rightarrow \bar{\alpha} \mid A \rightarrow \alpha \in P_i\}$ .

If  $\alpha = u$  with  $u \in V_T^*$ , then we have:

$$\{(A, v) \rightarrow v_1(X, v_2)\} \cup \bar{P}_i$$

where if  $u = vv'$ , for some  $v' \in V_T^*$ , then  $v_1 = v'$  and  $v_2 = \lambda$ , or otherwise, if  $v = uu'$ , for some  $u' \in V_T^*$ , then  $v_1 = \lambda$  and  $v_2 = u'$ , and  $\bar{P}_i$  is as above. These components will do the same as the component  $P_i$  of  $G$  under look-ahead  $v$ . It is taken into consideration that either the prefix  $v$  of  $u$  has already been generated by the scanning components and only the corresponding suffix must be produced in the first step, or the suffix of the scanned look-ahead string which is no part of  $u$  is stored in the new nonterminal which is now leftmost.

Note that the scanning components have nonempty look-aheads, the simulating components which rewrite some  $(A, v)$  with  $|v| < k$ , on the other hand, are to be used under empty look-ahead string.

(3) *Look-ahead shifting components.* For all  $u \in V_T^{\leq k}$ , we have:

$$\begin{aligned} & \{(X, u) \rightarrow (X, u)^{(1)}, \dots, (X, u)^{(m-1)} \rightarrow (X, u)^{(m-2)}, (X, u)^{(m-2)} \rightarrow \lambda\} \cup \\ & \{B \rightarrow (B, u) \mid B \in V_N \cup \bar{V}_T\}, \text{ and} \\ & \{(\bar{b}, u) \rightarrow (\bar{b}, u)^{(1)}, \dots, (\bar{b}, u)^{(m-1)} \rightarrow (\bar{b}, u)^{(m-2)}, (\bar{b}, u)^{(m-2)} \rightarrow b(X, \delta) \mid \bar{b} \in \bar{V}_T, \\ & \delta = \lambda \text{ if } u = \lambda, \text{ or } \delta = u' \text{ if } u = bu'\}. \end{aligned}$$

By these components the stored look-ahead string  $u$  is transferred to the next symbol to the right of the leftmost nonterminal symbol. If this next symbol is from  $\bar{V}_T$ , then the corresponding terminal symbol is generated and the rest of the look-ahead is shifted further, if it is from  $V_N$ , a look-ahead string of maximal possible length ( $k$ , in general) is supplemented to it with the help of the scanning components. Only then, the next simulation can be performed.

The comments given to the components constructed above show that the equalities  $L_{\text{sw}}(H, \mu) = L_{\text{sw}}(G, \mu)$ ,  $\mu \in \{t, =m \mid m \geq 2\}$ , hold, and as  $H$  satisfies the LL(1) property, the proof of is complete.  $\square$

From Theorem 5.5, and Theorem 5.9 we obtain the following corollary.

**Corollary 5.10** ([24]) Given a deterministic CD grammar system  $G$  satisfying the LL( $k$ ) condition,  $k \geq 1$ , and its strong lookup table for derivation mode  $=m$ ,  $m \geq 2$  with type of leftmostness sw. Then  $L_{\text{sw}}(G, =m) \in \text{dCD}_{\infty}\text{LL}(1)(=2, \text{sw})$ .

Now we present a decidable condition which a CD grammar system has to satisfy in order that the lookup table be effectively constructible.

**Definition 5.4** ([24]) A deterministic CD grammar system  $G = (V_N, V_T, S, P_1, \dots, P_n)$ ,  $n \geq 1$ , satisfies the *strong-LL( $k$ ) condition* for some  $k \geq 1$ , if for all  $i$ ,  $1 \leq i \leq n$ , and productions  $A \rightarrow \alpha \in \bigcup_{1 \leq i \leq n} P_i$ , the fact that  $A \rightarrow \alpha \in P_i$  implies that  $A \rightarrow \alpha \notin P_j$  for all  $j \neq i$ . Moreover, for all productions  $A \rightarrow \alpha$ ,  $A \rightarrow \beta \in \bigcup_{1 \leq i \leq n} P_i$ , such that  $\alpha \neq \beta$ , the condition

$$\text{FIRST}_k(\alpha \text{FOLLOW}_k(A)) \cap \text{FIRST}_k(\beta \text{FOLLOW}_k(A)) = \emptyset$$

holds with respect to the context-free grammar  $(V_N, V_T, S, \bigcup_{1 \leq i \leq n} P_i)$ .

Since the FIRST and FOLLOW sets for context-free grammars are effectively constructible, the strong-LL( $k$ ) condition for deterministic CD grammar systems is algorithmically decidable. Note also, that each of the CD grammar systems presented in Examples 5.1 and 5.2 satisfies the strong-LL( $k$ ) property, for the respective value of  $k$ .

For all context-free grammars  $(V_N, V_T, S, P)$  which satisfy the strong-LL( $k$ ) property, a strong lookup table  $M \subset (V_N \times V_T^{\leq k} \times P)$  can be effectively constructed as described, for example, in [3]. Thus, we can effectively construct a lookup table  $M_G$  also for any CD grammar system  $G = (V_N, V_T, S, P_1, P_2, \dots, P_n)$  satisfying the strong LL( $k$ ) property by constructing  $M_H$  for  $H = (V_N, V_T, S, \bigcup_{1 \leq i \leq n} P_i)$ , and then let  $(A, w, P_i) \in M_G$  if and only if  $(A, w, A \rightarrow \alpha) \in M_H$ , for some  $A \rightarrow \alpha \in P_i$ .



**Corollary 5.11 ([24])** Let  $\mu \in \{t, =m \mid m \geq 2\}$  and  $k \geq 1$ . If a deterministic CD grammar system  $G$  satisfies the strong-LL( $k$ ) property, then  $L_{sw}(G, \mu) \in \text{dCD}_n\text{LL}(1)(\mu, sw)$  and, moreover, both the corresponding deterministic CD grammar system satisfying the LL(1) condition and the strong lookup table for deterministic parsing can be effectively constructed.

## 5.4 Conclusion

Cooperating distributed grammar systems working in the  $t$ - and  $=m$ -modes of derivation,  $m \geq 2$ , have been restricted in a way such that, on the one hand, they maintain enough power in order to generate all context-free LL( $k$ ) languages, the languages  $L_1$ ,  $L_2$  and  $L_3$  of the concept of mildly context-sensitive grammars and even some non-semilinear language, but, on the other hand, there is an efficient parsing algorithm of  $O(n \cdot \log^2 n)$  time complexity.<sup>4</sup>

It is worth mentioning here, that the logarithm in the time bound of the algorithm is squared only because we carefully count bit operations in which reading and writing a search index  $n$  takes  $O(\log n)$  time. A uniform measurement of our algorithm would yield a time complexity of  $O(n \cdot \log n)$ .

The focus was on the development of the concept of an LL( $k$ ) condition which is appropriate for those systems, and of the parsing algorithm. The corresponding families of languages ( $\text{CD}_n\text{LL}(k)(\mu, \alpha)$ ,  $\mu \in \{t, =m \mid m \geq 1\}$ ,  $\alpha \in \{ss, sw\}$ ) need further investigations. The development of a construction method for a lookup table for CD grammar systems which are LL( $k$ ) but not strong LL( $k$ ) is of importance. Furthermore, the following decision problem might be of interest: Is it decidable whether a given CD grammar system is LL( $k$ ), for a given  $k$  or for any  $k$ ? Moreover, one could extend the research to other derivation modes. Similar investigations could also be led for further grammar formalisms such as matrix or programmed grammars. Finally, other restrictions like an appropriate LR( $k$ ) condition can be taken into consideration.

---

<sup>4</sup>The ability to generate a non-semilinear language, however, means that CD grammar systems satisfying the LL( $k$ ) condition do *not* define a new class of mildly context-sensitive languages.

# Bibliography

- [1] S. Ábrahám. Some questions of phrase-structure grammars I. *Comput. Linguistics* **4** (1965), 61–70.
- [2] A.V. Aho. Indexed grammars. An extension of context-free grammars. *Journal of the ACM* **15** (1968), 647–671.
- [3] A.V. Aho, J.D. Ullman. *The Theory of Parsing, Translation, and Compiling. Volume I: Parsing*. Prentice Hall, Englewood Cliffs, N.J., 1972.
- [4] A.V. Aho, J.D. Ullman. *The Theory of Parsing, Translation, and Compiling. Volume II: Compiling*. Prentice Hall, Englewood Cliffs, N.J., 1973.
- [5] A.W. Appel. *Modern Compiler Implementation in Java. Basic Techniques*. Cambridge University Press, 1998.
- [6] H. Bordihn. On some deterministic grammars with regulated rewriting. *Analele Universității București XXXIX-XL*(3), 35–48, 1990-1991.
- [7] H. Bordihn. Pure languages and the degree of nondeterminism. *Journal of Information Processing and Cybernetics (formerly: EIK)* **28**(5), 231–240, 1992.
- [8] H. Bordihn. A grammatical approach to the LBA problem. In Gh. Păun and A. Salomaa (editors), *New Trends in Formal Languages*, (LNCS 1218), 1–9. Springer, Berlin, Heidelberg, 1997.
- [9] H. Bordihn. Mildly context-sensitive grammars. In C. Martín-Vide, V. Mitrană, Gh. Păun (editors), *Formal Languages and Application, Studies in Fuzziness and Soft Computing 148*, 163–173. Springer, Berlin, Heidelberg, 2004.
- [10] H. Bordihn. On the number of components in cooperating distributed grammar systems. *Theoretical Computer Science* **330**, 195–204, 2005.
- [11] H. Bordihn, E. Csuhaĵ-Varjú, On competence and completeness in CD grammar systems, *Acta Cybernetica* 12 (1996) 347–360.
- [12] H. Bordihn, E. Csuhaĵ-Varjú, J. Dassow. CD grammar systems versus L systems. In Gh. Păun and A. Salomaa (editors), *Grammatical Models of Multi-Agent Systems* (Topics in Computer Mathematics 8), 18–32. Gordon and Breach Science, 1999.
- [13] H. Bordihn, J. Dassow, Gy. Vaszil. Grammar systems as language analyzers and recursively enumerable languages. In G. Ciobanu, Gh. Păun (editors), *Fundamentals of*

- Computation Theory, 12th International Symposium, FCT'99* (LNCS 1684), 136–147. Springer, Berlin, Heidelberg, 1999.
- [14] H. Bordihn, J. Dassow, Gy. Vaszil. Parallel communicating grammar systems as language analyzers. *Grammars* **3** (2000), 1–20.
- [15] H. Bordihn, H. Fernau. Accepting grammars and systems. Technical Report 9/94, Universität Karlsruhe, Fakultät für Informatik, 1994.
- [16] H. Bordihn, H. Fernau. Accepting grammars with regulation. *International Journal of Computer Mathematics* **53** (1994), 1–18.
- [17] H. Bordihn, H. Fernau. Accepting grammars and systems via context condition grammars. *Journal of Automata, Languages and Combinatorics* **1** (1996), 97–112.
- [18] H. Bordihn, H. Fernau, M. Holzer: On accepting pure Lindenmayer systems. *Fundamenta Informaticae* **38** (1999), 365–375.
- [19] H. Bordihn, H. Fernau, M. Holzer. Accepting pure grammars. *Publicationes Mathematicae* **60** (2002), 483–510.
- [20] H. Bordihn, M. Holzer. On the computational complexity of synchronized context-free languages. In C.S. Calude, K. Salomaa, S. Yu (editors), *Advances and Trends in Automata and Formal Languages, Journal of Universal Computer Science*, **8** (2002), 119–140.
- [21] H. Bordihn, M. Holzer. Programmed grammars and their relation to the LBA problem. *Acta Informatica* **43** (2006), 223–242.
- [22] H. Bordihn, Gy. Vaszil. CD grammar systems with  $LL(k)$  conditions. In E. Csuhaj-Varjú, Gy. Vaszil (editors), *Proceedings of Grammar Systems Week*, 95–112. MTA SZTAKI, Budapest, 2004.
- [23] H. Bordihn and Gy. Vaszil. On leftmost derivations in CD grammar systems. In R. Loos, Sz.Zs. Fazekas, C. Martín-Vide (editors), *1st International Conference on Language and Automata Theory and Applications, LATA'07*, Reports 35/07, 187–198. Universitat Rovira I Virgli, Tarragona, Spain, 2007.
- [24] H. Bordihn and Gy. Vaszil. Top-down deterministic parsing of languages generated by CD grammar systems. In E. Csuhaj-Varjú and Z. Ésik (editors), *Fundamentals of Computation Theory, 16th International Symposium, Budapest, Hungary, August 27-30, 2007, Proceedings*, (LNCS 4639), 113–124. Springer, Berlin, Heidelberg, 2007.
- [25] H. Bordihn and Gy. Vaszil. Leftmost derivations in CD grammar systems. *Submitted to Acta Informatica*, 2011.
- [26] J.L. Chen, C.W. Greider. Functional analysis of the pseudoknot structure in human telomerase RNA. *Proceedings of the Natural Academy of Sciences of the USA (PNAS)* **105** (2005), 8080–8085.
- [27] N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory* **2** (1956), 113–124.

- [28] T.H. Cormen, C.E. Leiserson, R.L. Rivest. *Introduction to Algorithms*. MIT Press and McGraw-Hill Book Company, 1990.
- [29] E. Csuhaj-Varjú. On grammars with local and global context conditions. *International Journal of Computer Mathematics* **47** (1993), 17–27.
- [30] E. Csuhaj-Varjú. Networks of Language Processors. *Bulletin of the EATCS* **63** (1997), 120–134.
- [31] E. Csuhaj-Varjú, J. Dassow. On cooperating distributed grammar systems. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)* **26** (1990), 49–63.
- [32] E. Csuhaj-Varjú, J. Dassow, J. Kelemen, Gh. Păun. *Grammar systems. A Grammatical Approach to Distribution and Cooperation*. Gordon and Breach, London, 1994.
- [33] E. Csuhaj-Varjú and Gy. Vaszil. On the computational completeness of context-free parallel communicating grammar systems. *Theoretical Computer Science* **215** 1-2 (1999), 349–358.
- [34] E. Csuhaj-Varjú and Gy. Vaszil. On context-free parallel communicating grammar systems: Synchronization, communication, and normal forms. *Theoretical Computer Science* **255** (2001), 511–538.
- [35] Ch. Culy. The complexity of the vocabulary of Bambara. *Ling. and Philosophy* **8** (1985), 345–351.
- [36] J. Dassow. Pure grammars with regulated rewriting. *Rev. Roumaine Math. Pures Appl.* **31** (1986), 657–666.
- [37] J. Dassow. A remark on limited 0L systems. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)* **24** (1988), 287–291.
- [38] J. Dassow, H. Fernau, Gh. Păun. On the leftmost derivation in matrix grammars. *Int. Journal of Foundations of Computer Science.* **10** (1999), 61–80.
- [39] J. Dassow and V. Mitrană. On the leftmost derivation in cooperating grammar systems. *Revue Roumaine de Mathématiques Pures et Appliquées*, **43** (1998), 361–374.
- [40] J. Dassow and Gh. Păun. Further remarks on pure grammars with regulated rewriting. *Rev. Roumaine Math. Pures Appl.* **31** (1986), 855–864.
- [41] J. Dassow, Gh. Păun. *Regulated Rewriting in Formal Language Theory*. Volume 18 of *EATCS Monographs in Theoretical Computer Science*. Springer, Berlin, Heidelberg, 1989.
- [42] J. Dassow and Gh. Păun. Cooperating/distributed grammar systems with registers. *Foundations of Control Engineering*, **15** (1990), 19–38.
- [43] J. Dassow, Gh. Păun, G. Rozenberg. Grammar systems. In [97], 155–213.
- [44] J. Dassow, Gh. Păun, A. Salomaa. Grammars with controlled derivations. In [97], 101–154.

- [45] S. Dumitrescu. Non-returning PC grammar systems can be simulated by returning systems. *Theoretical Computer Science* **165** (1996), 463-474.
- [46] H. Fernau. On function-limited Lindenmayer systems. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)* **27** (1991), 21-53.
- [47] H. Fernau. Membership for 1-limited ET0L languages is not decidable. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)* **30** (1994), 191-211.
- [48] H. Fernau. Remarks on adult languages of propagating systems with restricted parallelism. In G. Rozenberg and A. Salomaa, editors, *Developments in Language Theory (Turku, 1993)*, 90-101. Singapore: World Scientific, 1994.
- [49] H. Fernau. On unconditional transfer. In W. Penczek, A. Szalas (editors), *Proceedings of MFCS'96* (LNCS 1113), 348-359, Springer, Berlin Heidelberg, 1996.
- [50] H. Fernau. Remarks on regulated limited ET0L systems and regulated context-free grammars. *Theoretical Computer Science* **194** (1998), 35-55.
- [51] H. Fernau. Regulated grammars under leftmost derivation. *Grammars* **3** (2000), 37-62.
- [52] H. Fernau, H. Bordihn. Remarks on accepting parallel systems. *International Journal of Computer Mathematics* **56** (1995), 51-67.
- [53] H. Fernau, M. Holzer, H. Bordihn. Accepting multi-agent systems: The case of cooperating distributed grammar systems. *Computers and Artificial Intelligence* **15**(1996), 123-139.
- [54] R.W. Floyd. On the non-existence of a phrase structure grammar for ALGOL 60. *Communications of the ACM* **5** (1962), 483-484.
- [55] M. Frings. Systeme mit eingeschränkter paralleler Ersetzung. Master's thesis, TU Braunschweig, D-3300 Braunschweig, 1985.
- [56] I. Friš. Grammars with partial orderings of the rules. *Information and Control (now Information and Computation)*, **12** (1968), 415-425.
- [57] A. Gabrielian. Pure grammars and pure languages. Technical Report CSRR 2027, Univ. of Waterloo, Dept. of Comp. Sci., 1970.
- [58] S. Ginsburg, E.H. Spanier. Control sets on grammars. *Mathematical Systems Theory* **2** (1968), 159-177.
- [59] S. Greibach, J.E. Hopcroft. Scattered context grammars. *Journal of Computer and System Sciences* **3** (1969), 233-247.
- [60] D. Hauschildt and M. Jantzen. Petri net algorithms in the theory of matrix grammars. *Acta Informatica* **31** (1994), 719-728.
- [61] T. Head, Gh. Păun, D. Pixton. Language theory and molecular genetics. In [97], 295-360.
- [62] G.T. Herman, G. Rozenberg. *Developmental Systems and Languages*. North-Holland, Amsterdam, 1975.

- [63] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Reading (MA): Addison-Wesley, 1979.
- [64] J. Hromkovič, On the communication complexity of distributive language generation. In: J. Dassow, G. Rozenberg and A. Salomaa (eds.), *Developments in Language Theory II*, 237–246. World Scientific, Singapore, 1995.
- [65] A.K. Joshi. Tree adjoining grammars: How much context-sensitivity is necessary for characterizing structural descriptions. In: D. Dowty, L. Karttunen, A. Zwicky (editors), *Natural Language Parsing: Psychological, Computational and Theoretical Perspectives*, 206–250. Cambridge University Press, New York, 1985.
- [66] A.K. Joshi, L.S. Levi, M. Takahashi. Tree adjunct grammars. *J. Comput. System Sci.* **10** (1975), 136–163.
- [67] A.K. Joshi, Y. Schabes. Tree-adjoining grammars. In [98], 69–123.
- [68] H. Jürgensen, K. Salomaa. Block-synchronized context-free grammars. In D.Z. Du and J.I. Ko (editors), *Advances in Algorithms, Languages, and Complexity*, 111–137. Kluwer, 1997.
- [69] S.Y. Kuroda. Classes of languages and linear bounded automata. *Information and Control* **7** (1964), 207–223.
- [70] N. Mandache. On the computational power of context-free PC grammar systems. *Theoretical Computer Science* **237** (2000), 135–148.
- [71] A. Mateescu and A. Salomaa. Aspects of classical language theory. In [97], 175–251.
- [72] H.A. Maurer, A.K. Salomaa, and D. Wood. Pure grammars. *Information and Control* **44** (1980), 47–72.
- [73] *MDA Model Driven Architecture*. <http://www.omg.org/mda>
- [74] A. Meduna. A trivial method of characterizing the family of recursively enumerable languages by scattered context grammars. *EATCS Bulletin* **56** (1995), 104–106.
- [75] R. Meersman, G. Rozenberg. Cooperating grammar systems. *Proc. MFCS'78* (LNCS 64), 364–374, Springer, Berlin, Heidelberg, 1978.
- [76] V. Mihalache. Accepting cooperating distributed grammar systems with terminal derivation. *EATCS Bulletin* **61** (1997), 80–84.
- [77] V. Mihalache, V. Mitrana. Deterministic cooperating distributed grammar systems. In Gh. Păun and A. Salomaa (editors), *New Trends in Formal Languages*, (LNCS 1218), 137–149. Springer, Berlin, Heidelberg, 1997.
- [78] D. Milgram, A. Rosenfeld. A note on scattered context grammars. *information Processing Letters* **1** (1971), 47–50.
- [79] V. Mitrana. Hybrid cooperating/distributed grammar systems. *Computers and Artificial Intelligence* **12** (1993), 83–88.

- [80] V. Mitrana. Parsability approaches in CD grammar systems. In R. Freund, A. Kelemenová, editors, *Proceedings of the International Workshop Grammar Systems 2000*, 165–185. Silesian University, Opava, 2000.
- [81] B. Monien. On the LBA problem. In F. Gécseg (editor) *Proceeding of the International Conference Fundamentals on Computation Theory* (LNCS 117), 265–280. Springer, Berlin, Heidelberg, 1981.
- [82] E. Navrátil. Context-free grammars with regular conditions. *Kybernetika* **6** (1970), 118–126.
- [83] K. Ogasawara, S. Kobayashi. Stochastically approximating tree grammars by regular grammars and its application to faster ncRNA family annotation. In R. Loos, Sz.Zs. Fazekas, C. Martín-Vide (editors), *1st International Conference on Language and Automata Theory and Applications, LATA '07*, Reports 35/07, 187–198. Universitat Rovira I Virgli, Tarragona, Spain, 2007.
- [84] B.H. Partee, A. ter Meulen, R. Wall. *Mathematical Methods in Linguistics*. Kluwer, 1993.
- [85] Gh. Păun. On the generative capacity of conditional grammars. *Information and Control (now Information and Computation)* **43** (1979), 178–186.
- [86] Gh. Păun. A variant of random context grammars: semi-conditional grammars. *Theoretical Computer Science* **41** (1985), 1–17.
- [87] Gh. Păun. On the synchronization in parallel communicating grammar systems. *Acta Informatica* **30** (1993), 351–367.
- [88] Gh. Păun. On the generative capacity of hybrid CD grammar systems. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)* **30** (1994), 231–244.
- [89] Gh. Păun. Grammar systems: a grammatical approach to distribution and cooperation. In *Automata, Languages and Programming; 22nd International Colloquium, ICALP'95, Szeged, Hungary*, (LNCS 944), 429–443, Springer, Berlin, Heidelberg, 1995.
- [90] Gh. Păun, G. Rozenberg, A. Salomaa. *DNA Computing. New Computing Paradigms*. Springer, Berlin, Heidelberg, 1998.
- [91] Gh. Păun, L. Sântean. Parallel communicating grammar systems: the regular case. *Ann. Univ. Buc., Ser. Matem.-Inform.* **38** (1989), 55–63.
- [92] G. Pullum, G. Gazdar. Natural languages and context-free languages. *Linguistics and Philosophy* **4** (1982), 471–504.
- [93] D.J. Rosenkrantz. Programmed grammars and classes of formal languages. *Journal of the Association for Computing Machinery* **16** (1969), 107–131.
- [94] D.J. Rosenkrantz, R.E. Stearns. Properties of deterministic top-down grammars. *Information and Control* **17** (1970), 226–256.
- [95] G. Rozenberg. Extension of tabled 0L-systems and languages. *International Journal of Computer and Information Sciences* **2** (1973), 311–336.

- [96] G. Rozenberg, A. Salomaa. *The Mathematical Theory of L Systems*. Academic Press, New York, 1980.
- [97] G. Rozenberg, A. Salomaa. *Handbook of Formal Languages. Vol.2, Linear Modelling: Background and Application*. Springer, Berlin, Heidelberg, 1997.
- [98] G. Rozenberg, A. Salomaa. *Handbook of Formal Languages. Vol.3, Beyond Words*. Springer, Berlin, Heidelberg, 1997.
- [99] A. Salomaa. *Formal Languages*. Academic Press, New York, 1973.
- [100] K. Salomaa. Hierarchy of  $k$ -context-free languages. *International Journal of Computer Mathematics* **26** (1989) 69–90, 193–205.
- [101] D.C. Schmidt. Model-driven engineering. *IEEE Computer* **39** (2006), 25–31.
- [102] R. Sethi. *Programming Languages. Concepts & Constructs*. Addison Wesley, 1996.
- [103] S.M. Shieber. Evidence against the context-freeness of natural language. *Linguistics and Philosophy* **8** (1985), 333–343.
- [104] K. Sikkil, A. Nijholt. Parsing of context-free languages. In [97], 61–100.
- [105] R. Siromoney, K. Krithivasan. Parallel context-free grammars. *Information and Control* **24** (1974).
- [106] S.H. von Solms. Some notes on ETOL-languages. *International Journal of Computer Mathematics* **5** (1976), 285–296.
- [107] F.L. Țiplea, C. Ene, C.M. Ionescu and O. Procopiu. Some decision problems for parallel communicating grammar systems. *Theoretical Computer Science* **134** (1994), 365–385.
- [108] Gy. Vaszil. On simulating non-returning PC grammar systems with returning systems. *Theoretical Computer Science* **209** (1998), 319–329.
- [109] Gy. Vaszil. *Investigations on parallel communicating grammar systems*. PhD thesis, Eötvös Loránd University, Budapest, 2000.
- [110] K. Vijay-Shanker, D.J. Weir. The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory* **87** (1994), 511–546.
- [111] P.M.B. Vitányi. Context sensitive table Lindenmayer languages and a relation to the LBA problem. *Information and Control* **33** (1977), 217–226.
- [112] A.P.J. van der Walt. Random context grammars. Proc. Symp. on Formal Languages, Oberwolfach, 1970.
- [113] D. Wätjen.  $k$ -limited 0L systems and languages. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)* **24** (1988), 267–285.
- [114] D. Wätjen. A weak iteration theorem for  $k$ -limited EOL systems. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)* **28** (1992), 37–40.



- [115] D. Wätjen, E. Unruh. On extended  $k$ -uniformly-limited T0L systems and languages. *J. Inf. Process. Cybern. EIK (formerly Elektron. Inf.verarb. Kybern.)* **26** (1990), 283–299.
- [116] D. Wood. Bicolored digraph grammar systems. *RAIRO, Rech. Oper.* **R-1** (1973), 45–50.
- [117] D. Wood. *Theory of Computation*. John Wiley & Sons, New York, 1987.
- [118] G. Zetsche. On erasing productions in random context grammars. In S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide and P. G. Spirakis (editors), *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II* (LNCS 6199), 175–186. Springer, Berlin, Heidelberg, 2010.
- [119] G. Zetsche. A sufficient condition for erasing productions to be avoidable. In G. Mauri and A. Leporati (editors), *Developments in Language Theory, 15th International Conference, DLT 2011, Milan, Italy, July 19-22, 2011, Proceedings* (LNCS 6795), 452–463. Springer, Berlin, Heidelberg, 2011.
- [120] G. Zetsche. Towards understanding the generative capacity of erasing rules in matrix grammars. *International Journal of Foundations of Computer Science* **22** (2011), 411–426.