Controlled Bidirectional Grammars

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Universiteit Twente, op gezag van de rector magnificus prof.dr.ir. J.H.A. de Smit volgens besluit van het College van Dekanen in het openbaar te verdedigen op vrijdag 31 augustus 1990 te 16.00 uur

door

Jan Anne Hogendorp

geboren op 22 juni 1958 te Wommels

Dit proefschrift is goedgekeurd door

Prof.dr.ir. A. Nijholt	promotor,
Prof.dr.ir. L.A.M. Verbeek	promotor, en
Dr.ir. P.R.J. Asveld	assistent-promotor.

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Hogendorp, Jan Anne

Controlled bidirectional grammars / Jan Anne Hogendorp. -
[S.l. : s.n.]. – Ill.
Proefschrift Enschede. – Met lit. opg. – Met samenvatting
in het Fries en Nederlands.
ISBN 90-9003620-2
SISO 805 UDC 801.5
Trefw.: grammatica / formele talen.

print: FEBO drukkerij Enschede

oan myn âlden

Preface

Most grammar models contain a set of rewriting rules of some kind, e.g. string-rewriting rules or term-rewriting rules. These rules are applied in one direction only. Usually, in a grammar two or more alphabets are distinguished. In general the language generated by a grammar is defined as the set of words (over some distinguished alphabet) that can be obtained by applying the rewriting rules in a unidirectional way, starting from some designated symbol, the start symbol.

In a Thue system the rules are applied in both directions. However, a Thue system possesses a single alphabet. And there is no standard definition of the language associated with a Thue system.

Therefore it seemed interesting to study grammar models in which the rewriting rules can be applied in both directions, while maintaining the classical language definition. In this thesis we investigate such bidirectional variants of context-free grammars (Chapters II-IV) and (extended) linear basic grammars (Chapters V-VI). We focus attention to closure properties, grammatical transformations (that result in a weak Chomsky normal form), parsing algorithms, and determining the language generating power of these grammatical models.

Another motive for studying bidirectionality in grammar models stems from NTS or nonterminal separating grammars – one of the most well-known types of grammar that satisfies the "disjunct syntactic categories property". NTS grammars are based on context-free grammars and on the concept of bidirectionality. It is straightforward to extend the NTS definition to macro grammars; cf. Appendix A.

Our main results are in Chapters II-VI. They are preceded by an Introduction (Chapter I) and followed by Chapter VII which consists of concluding remarks and a few areas of computer science in which our results may be applied.

The subject of this thesis is rather new and – in our opinion – it constitutes an interesting way of specifying languages. Therefore, there are at this moment more open questions than results. Particularly, the need of applications is conspicuous.

Acknowledgements

I am indebted to Peter Asveld for his contribution to both content and form of this thesis. His criticism, knowledge and ideas were of great value to me. Thanks are due to Anton Nijholt and Leo Verbeek for reading the manuscript and for their constructive remarks. I also want to thank Henk Alblas for his participation in the *begeleidingscommissie*.

I am grateful to Rieks op den Akker for some discussions on a few topics in this thesis. Thanks also to the members of the *vakgroepen* TIF and SETI for a nice working atmosphere. In this respect I like to mention Charlotte Bijron, Maarten Fokkinga, Thérèse ter Heide-Noll, Alice Hoogvliet-Haverkate, Jan Kuper and Joke Lammerink, and, last but not least, my roommates Jan Molenkamp, Paul Oude Luttighuis and Gert Veldhuijzen van Zanten.

The research resulting in this thesis has been made possible by a grant (nr. 612-316-012) from the Netherlands Organization of Scientific Research (NWO), to which I am grateful. I also thank the Department of Computer Science of the University of Twente for its additional support.

Contents

CHAPTER I

Introduction

1. Specifying Formal Languages

Formal language theory deals with languages and with devices which represent languages in a finite fashion. A *language* is a set of words over some alphabet. By an *alphabet* we mean a finite set of symbols. A *word* over an alphabet Σ is an element of the free monoid Σ^* generated by Σ under concatenation, and with the empty word, denoted by λ, as its two-sided identity. A word is also called a *string*.

We can distinguish two methods to specify a language. The first method describes how words ought to look like in order to belong to the language. Such a *property specification* can be stated in first-order predicate logic and will be algebraic and "static" of nature. For example, we can describe the language of "doubled sentences" or "copies" over the alphabet Σ by $L_0 = \{x \mid x = ww,\ w \in \Sigma^*\}$, which can be shortened to $\{ww \mid w \in \Sigma^*\}$.

However, formal language theory is much more concerned with another, second method of specifying languages. Instead of giving a description, one can introduce a device that determines a language in an active way. The active part of such a device uses rewriting rules. A rewriting rule, which is an ordered pair of words, tells us how to obtain one string from another. The rewriting of a given word to another by a rewriting rule is called the application of that rewriting rule to the given word. For example, the rewriting rule (aa, ab) rewrites *aaab* into *abab* or *aabb*. A rewriting system over some alphabet is a set of rewriting rules. Thus a rewriting system is in fact a relation over the set of words over some alphabet.

We can define a language by means of a rewriting system either in a generating or in an accepting way. In the resulting generating device we start from a given set of initial words. The language generated by such a device consists of all words which can be obtained by the successive application of zero or more rewriting rules to some initial word. On the other hand, an accepting device has, apart from rewriting rules, a set of halting words. The language accepted – or recognized – by an accepting device is the set of all words which can yield some halting word by successively

applying (zero or more) rewriting rules.

Intuitively, these concepts of generating and accepting seem to be dual to each other. Indeed, these concepts can be defined without much effort such that the language associated with an accepting device equals the language produced by a corresponding generating device of which the set of initial words is equal to the set of accepting words. This generating device has as its set of rewriting rules the (set-theoretical) converse of the set of rewriting rules of the accepting device. (Remember that a rewriting system over Σ is some relation over Σ^*). Then the dual proposition, replacing "accepting" by "generating" and "initial" by "halting" and vice versa, also holds.

Rewriting systems are applied in many areas of computer science. For instance programming languages, parsing theory, specification languages etcetera. In most of these cases the unidirectional character of rewriting gives sufficient power to be useful in these areas. For instance, there is a huge amount of theory on phrase-structure grammars, finite state automata, Turing machines, etcetera, each of which can be considered as a particular type of rewriting system. However, in fields like theorem proving and program transformation there is a need to use rewriting rules in a reversed way too. This can be obtained by joining together a set of rewriting rules and its converse. The resulting system is a Thue system [Thu]. In fact, in a Thue system it is not necessary to add for each rewriting rule (u,v) its inverse rewriting rule (v,u) because the latter is assumed to be available in a Thue system. Therefore, a rewriting system is sometimes called a semi-Thue system. So, in a Thue system T the rewriting of a string w is performed by applying a rewriting rule (u,v) in T or its inverse (v,u).

A Thue system induces an equivalence relation as follows. Two words are equivalent by the Thue system T if they can be rewritten into each other by the rewriting rules of T. Actually, this is a congruence relation with respect to concatenation. A word z is a reduct of a word x in case z is obtained from x by applying only length-decreasing rewriting rules, i.e., rewriting rules (u,v) in T with $|u| > |v|$. A word is irreducible if no length-decreasing rewriting rule is applicable to it. A Thue system T is called finite if T is a finite set.

In the theory of Thue systems a traditional topic is the so-called word problem. This is the problem of deciding whether or not two words over the alphabet are equivalent with respect to the rewriting rules of the Thue system. The study of Thue systems is also motivated by regarding Thue systems as an instrument to define formal languages. A Thue system has the Church-Rosser property if each two equivalent words have a common

unique irreducible reduct. Thue systems obeying the Church-Rosser property received – and still receive – a lot of attention. The main reason is that congruence classes of finite Church-Rosser Thue systems – i.e., finite Thue systems which have the Church-Rosser property – are languages recognizable by a deterministic linear-bounded automaton. This is the approach taken by Nivat and others in the 1960s and 1970s; cf. [Niv, Ber].

If we define languages by Thue systems analogously to the way in which we define languages by rewriting systems, then we cannot distinguish between the generating and the accepting way of defining languages, due to the bidirectional character of Thue systems. However, in this thesis we study Thue systems of a special kind, which we call bidirectional grammars. A bidirectional grammar is a generating device obtained by taking the set of productions of a context-free grammar as the defining set of rules for a Thue system. Then we define languages analogously to the case of phrase-structure grammars; i.e., we derive sentential forms starting from an initial symbol (or a set of initial words) by applying productions and their corresponding inverse productions, called reductions. The generated language is defined as the set of sentential forms over a terminal alphabet.

Bidirectional grammars happen to be powerful generating devices. Therefore we restrict their power in a natural way. Essentially, this restriction consists of two parts. First, we use a control language over the rules – i.e., over the productions and reductions – and secondly, we attach a selection mechanism to a bidirectional grammar such that a rule can be applied to at most one substring of a sentential form. The introduction of a control language will in fact increase the generating power of bidirectional grammars. However, it also turns out to be an enormous help in establishing various theorems concerning (controlled) bidirectional grammars. In addition, it emphasizes the generative character of bidirectional grammars.

The remaining part of this chapter is organized as follows. In Section 2 some technical preliminaries are introduced, in order to make this thesis self contained. Most concepts introduced in this section such as rewriting systems, Thue systems, and macro grammars, are well known, and they are recalled here to establish our notation. However, the concept of NTS grammar (or nonterminal separating grammar) may need some more attention.

In Section 3 we introduce the concept of regularly controlled bidirectional grammar, based on either context-free or on macro grammars. In Section 4 Chapters II–VII are outlined.

2. Preliminaries

In this section we first consider rewriting systems and Chomsky grammars (§2.1). Then we recall the basic concepts related to Thue systems (§2.2). Finally, we discuss nonterminal separating – or NTS – grammars (§2.3) and macro grammars (§2.4) in somewhat more detail.

2.1. Rewriting Systems

Definition 2.1.1. A *rewriting system R* on the alphabet Ξ is a set of *rewriting rules*. A rewriting rule is an ordered pair (u,v) in $\Xi^*\times\Xi^*$. A rewriting rule or *production* (u,v) acts in one direction only, i.e., an occurrence of u in a string may be rewritten to v but not vice versa. The derivation relation $\Rightarrow_R^*$ (or $\Rightarrow^*$ if R is known from the context) is the transitive and reflexive closure of the *single-step* or *direct* derivation relation $\Rightarrow_R$ (or $\Rightarrow$) defined by

$$\Rightarrow_R = \{(x,y)\in\Xi^*\times\Xi^* \mid \exists\,(u,v)\in R,\ \exists\,w_1,w_2\in\Xi^* \bullet x = w_1uw_2 \wedge y = w_1vw_2\}.$$

□

A rewriting system can be used in two ways. One way is to define a language to be the set consisting of all words over a given alphabet which can be rewritten to a given word (or to a member of a given set of words). A word may initially be concatenated with some additional words before the actual rewriting starts; cf. Example 2.1.3. A device of this kind is called a *recognition device*. The other, dual, method is to define a language by the set of all words over a given alphabet Σ ($\Sigma\subseteq\Xi$) that can be obtained by starting from a given word (or a given set of words) and then applying zero or more rewriting rules. This kind of device is called a *generative device* or *grammar*. Each string obtainable from the starting word (or set of words) is called a *sentential form*. By a *sentence* we denote a sentential form in which only symbols from Σ occur. This approach resulted, among others, in the notion of phrase-structure grammar and related grammatical models by Chomsky in [Cho56, Cho59].

Definition 2.1.2. A *phrase-structure grammar G* is a 4-tuple (V,Σ,P,S), where

- V – the *vocabulary* – and Σ – the *terminal alphabet* – are finite alphabets with $\Sigma\subseteq V$. The elements of $V-\Sigma$ and Σ are called *nonterminals* and *terminals*, respectively.
- The *start symbol* S is an element of $V-\Sigma$.
- P is a finite set of ordered pairs in $(V^*-\Sigma^*)\times V^*$. Elements of P are called *productions*.*

* Productions are also called *rules*. But in the sequel we will use the word “rule” in a

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

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\},$$

where $\Rightarrow^*$ is the derivation relation associated with the rewriting system P on V. It is a well-known fact that the family of languages generated by phrase-structure grammars equals the family of recursively enumerable languages. □

Example 2.1.3. A finite state automaton $M = (Q, \Sigma, \delta, q_0, F)$, where

- Q is the finite set of states,
- Σ is the input alphabet,
- q_0 is the initial state ($q_0 \in Q$),
- F is the set of accepting states ($F \subseteq Q$),
- $\delta : Q \times \Sigma \to Q$ is transition function,

is usually defined as a machine that accepts the language

$$L(M) = \{w \in \Sigma^* \mid \delta'(q_0, w) \in F\},$$

where $\delta' : Q \times \Sigma^* \to Q$ is the extension of δ defined for all q in Q by

$$\delta'(q, \lambda) = q,$$

$$\delta'(q, a) = \delta(q, a),$$

$$\delta'(q, as) = \delta'(\delta(q, a), s),$$

with $a \in \Sigma$ and $s \in \Sigma^*$; cf. [Har] for details. However, M may also be considered as a rewriting system; particularly, as a recognition device. Then the language *accepted* by M is defined by

$$L(M) = \{w \in \Sigma^* \mid q_0 w \Rightarrow^* q,\ q \in F\},$$

where $\Rightarrow^*$ is the derivation relation associated with the rewriting system P on $Q \cup \Sigma$. The set of rewriting rules P is formed by

$$\{(pa, q) \mid \delta(p, a) = q,\ p, q \in Q,\ a \in \Sigma\}.$$ □

Example 2.1.4. The dual concept of a phrase-structure grammar is the *analytical grammar*, introduced by Salomaa [Sal73], which is another example of a recognition device. It is a 4-tuple (V, Σ, P, S) which differs from the 4-tuple defined in Definition 2.1.2 in the definition of P, which now is a finite subset of $V^* \times (V^* - \Sigma^*)$. An analytical grammar $G = (V, \Sigma, P, S)$ possesses an underlying rewriting system P on V which induces the relations $\Rightarrow$, i.e., one-step derivation and $\Rightarrow^*$, the reflexive and transitive closure of

---

different way.

$\Rightarrow$. The language $L(G)$ recognized by the analytical grammar G is defined by

$$L(G) = \{w \in \Sigma^* \mid w \Rightarrow^* S\}.$$

It is easy to show [Sal73] that phrase-structure grammars and analytical grammars are equivalent with respect to descriptive power, i.e., for each phrase-structure grammar G_0 we can find an analytical grammar G_1 such that $L(G_0) = L(G_1)$ and vice versa. □

Example 2.1.5. A *context-sensitive grammar* is a phrase-structure grammar $G = (V, \Sigma, P, S)$ in which each production is of the form $(\alpha A \beta, \alpha u \beta)$, where $\alpha, \beta \in V^*$, A in $V - \Sigma$ and $u \in V^+$ with the possible exception of the production $S \rightarrow \lambda$. However, if this production does occur in P, the symbol S does not occur in the right-hand side of any production in P. A language L_0 is called context-sensitive if there exists a context-sensitive grammar G such that $L_0 = L(G)$. We denote the family of context-sensitive languages by *CSL*. □

2.2. Thue Systems

Rewriting systems, and consequently the derived generative and analytical grammars, act into one direction, i.e., given a rewriting rule (u,v) we cannot rewrite a substring v of a word w to u, unless the rewriting rule (v,u) is an explicit element of the rewriting system too. And in case of generative and analytical grammars, adding such reversed rewriting rules is simply not allowed for each rule, due to the definition of their respective set of rewriting rules. The extension of such *unidirectional* devices to their corresponding *bidirectional* variants is a natural one, and in case of rewriting systems this leads to the concept of the well-known Thue system, named after the Norwegian mathematician and logician Axel Thue, who studied such systems at the beginning of the twentieth century [Thu]. Thue systems were one of the first known systems in which rewriting of strings of symbols was the main objective of research, long before the advent of Chomsky's grammatical model.

Definition 2.2.1. A *Thue system T* on the alphabet Ξ is a set of ordered pairs (u,v) of strings over Ξ. If Ξ is known from the context, we can denote a Thue system by its set of rewriting rules T. We can rewrite a string w by (u,v) in T if either u or v occurs in w and then the result of rewriting w is the string w' obtained from w by replacing an occurrence of the string u [or v, respectively] by the string v [u, respectively]. Then we write $w \Leftrightarrow_T w'$. □

Remark that $\Leftrightarrow_T$ is a relation on Ξ^*. Moreover, note that a rewriting system R on Ξ induces a Thue system on Ξ, in the sense that the rewriting system $R \cup \overline{R}$ on Ξ, where $\overline{R}$ is defined by $\overline{R} = \{(v,u) \mid (u,v) \in R\}$, (or, in

other words, $\overline{R}$ is the converse of R) is strongly equivalent to the Thue system R on Ξ. A rewriting system R is strongly equivalent to a Thue system T if

(i) for each rewriting rule (u,v) in T both (u,v) as well as (v,u) is in R, and

(ii) for each rewriting rule (u,v) in R either (u,v) or (v,u) is in T.

The alternative name of *semi-Thue system* for a rewriting system stems from this observation. In turn, a Thue system T on Ξ induces two rewriting systems T and $\overline{T}$ (both on Ξ). Observe that the derivation relation of the rewriting system $\overline{T}$, denoted by $\Leftarrow_T^*$, is the converse of the derivation relation $\Rightarrow_T^*$ of the rewriting system T. In addition, by $\Leftrightarrow_T^*$ (or $\Leftrightarrow^*$ when T is understood) we denote the derivation relation of the Thue system T. The relation $\Leftrightarrow_T^*$ is the reflexive and transitive closure of $\Leftrightarrow_T$. Note that $\Leftrightarrow_T^*$ equals the corresponding relation for $\overline{T}$. This derivation relation $\Leftrightarrow_T^*$ is a congruence relation on Ξ^* with respect to concatenation. We denote the congruence class (modulo T) of a word x by

$$[x]_T = \{w \in \Xi^* \mid x \Leftrightarrow_T^* w\}.$$

A rewriting rule (u,v) in T is called *length-decreasing* if $|u| > |v|$. Define the set of *descendants* of a word x by

$$\Delta_T^*(x) = \{w \in \Xi^* \mid x \Rightarrow_T^* w\}.$$

For a language $L_0 \subseteq \Xi^*$, we define $\Delta_T^*(L_0) = \cup\{\Delta_T^*(x) \mid x \in L_0\}$. Define the relation $\rightarrow_T$ on $\Xi^* \times \Xi^*$ by $x \rightarrow_T y$ if $x \Leftrightarrow_T y$ and $|x| > |y|$. Then x is *irreducible* (modulo T), if there is no y such that $x \rightarrow_T y$. The set $IRR(T)$ denotes the set of irreducible words over Ξ by T. The domain and range of a Thue system are defined by $dom(T) = \{u \mid \exists\, v \,.\, (u,v) \in T\}$ and $range(T) = \{v \mid \exists\, v \,.\, (u,v) \in T\}$, respectively.

A Thue system T is called *monadic*, if each (u,v) in T is length-decreasing and $range(T) \subseteq \Xi \cup \{\lambda\}$, i.e., $|v| \leq 1$. A monadic Thue system T is *special* if for each rule (u,v) in T, we have that $v = \lambda$.

Example 2.2.2. Consider the Thue system $T = \{(baa, ab)\}$ on $\{a,b\}$. Let L_0 be the regular set $\{b\}\{a\}^*$. Then the descendants of L_0 by the rewriting system T are given by

$$\Delta_T^*(L_0) = \{a^n b a^{m-2n} \mid 0 \leq 2n \leq m\},$$

and the irreducible words of the Thue system T are given by the set $IRR(T) = \{a\}^*\{b, ba\}^*$. Then we have

$$\Delta_T^*(L_0) \cap IRR(T) = \{a^n b,\ a^n ba \mid n \geq 0\}. \qquad \square$$

Nowadays, Thue systems still obtain widely attention from computer scientists, algebraists and logicians; cf. [Boo87] for an overview. Thue systems can be used in various ways to define formal languages; cf. [Boo82, BooJanWra, McNNarOtt].

In a bidirectional grammar based on a (well-known) unidirectional grammar of some type, the set of rewriting rules is also formed by the union of the set of productions P of the unidirectional grammar (called the *underlying* grammar) and its converse $\overline{P}$. In the sequel we call an element of $\overline{P}$ a *reduction* and an element of $P \cup \overline{P}$ a *rule*. The extension of the concept of phrase-structure grammar (or, equivalently, analytical grammar) to the bidirectional case has obtained little attention. This can be explained by the fact that even restricted subclasses of the set of phrase-structure grammars give rise to a dramatic growth of generating power, when extented in the bidirectional way. As an example we consider the case of context-free grammars.

Definition 2.2.3. A *context-free grammar* is a phrase-structure grammar $G = (V, \Sigma, P, S)$ which obeys the additional restriction $P \subseteq (V - \Sigma) \times V^*$. A production of the form (A, w), with $A \in V - \Sigma$ and $w \in V^*$, is called a context-free production. A *context-free grammar with initial set* M is like a context-free grammar a 4-tuple $G = (V, \Sigma, P, M)$, where M is a language. The language generated by the context-free grammar with initial set M, $G = (V, \Sigma, P, M)$ is defined by

$$L(G,M) = \{w \in \Sigma^* \mid \exists\, \alpha \in M \,.\, \alpha \Rightarrow^* w\}.$$

Note that we can also define $L(G)$ by $L(G) = L(G, \{S\})$. In addition, *CFL* denotes the family of context-free languages. □

In the sequel we will apply the convention of denoting a rewriting rule (u, v) in a rewriting system R by $u \rightarrow_R v$. Moreover, a rewriting rule (u, v) in a Thue system will T be denoted by $u \leftrightarrow_T v$. As usual, the subscripts R and T can be omitted if they are known from the context.

Example 2.2.4. Consider the bidirectional context-free grammar $G = (V, \Sigma, P, S)$, where $V = \{A, B, D, E, S, a, b, c\}$, $\Sigma = \{a, b, c\}$, and P consists of

$$\begin{aligned} \pi_0 &= S \rightarrow abc, & \pi_4 &= B \rightarrow bDbc, \\ \pi_1 &= S \rightarrow abDSc, & \pi_5 &= B \rightarrow bbc, \\ \pi_2 &= A \rightarrow bDa, & \pi_6 &= E \rightarrow bDbb, \\ \pi_3 &= A \rightarrow abD, & \pi_7 &= E \rightarrow bbb. \end{aligned}$$

We note that, when considered as a unidirectional context-free grammar, G is not reduced, i.e., there are nonterminals (viz. A, B, and E) which will never occur in some unidirectional derivation from S, and there are also

nonterminals (viz. A and D) for which there is no derivation which yields a terminal string. In fact, the language generated by the unidirectional grammar G is equal to $\{abc\}$.

We will show that G, considered as a bidirectional context-free grammar, generates $\{a^n b^n c^n \mid n \geq 1\}$. First, a sentential form $(\underline{abD})^n abcc^n$ $(n \geq 0)$ is generated. To this string are applicable the sequences $\bar{\pi}_2\pi_3$ and $\bar{\pi}_0\pi_1$. By $\bar{\pi}_2\pi_3$ a terminal b is moved to the right side of a terminal a. If $\bar{\pi}_4$ becomes applicable, followed by π_5, then the second terminal b from the right side of the sentential form has been put at its right position. In addition, the sequence $\bar{\pi}_6\pi_7$ becomes applicable in case a terminal b has been moved by several applications of sequences $\bar{\pi}_2\,\pi_3$ to the right side, until it encounters – only separated by a nonterminal D – his colleague terminals b which are already at their right position. For instance, a derivation of $a^4b^4c^4$ may be performed as follows.

$$S \Rightarrow^{\pi_1^2} abDabDScc \Rightarrow^{\pi_0} abDabDabccc \Rightarrow^{\bar{\pi}_2\pi_3} aabDbDabccc$$
$$\Rightarrow^{\bar{\pi}_0} aabDbDScc \Rightarrow^{\pi_1} aabDbDabDSccc \Rightarrow^{\pi_0} aabDbDabDabcccc$$
$$\Rightarrow^{\bar{\pi}_2\pi_3} aabDbDaabDbcccc \Rightarrow^{\bar{\pi}_4\pi_5} aabDbDaabbcccc$$
$$\Rightarrow^{(\bar{\pi}_2\pi_3)^4} aaaabDbDbbcccc \Rightarrow^{(\bar{\pi}_6\pi_7)^2} aaaabbbbcccc \qquad \square$$

2.3. NTS Grammars

In [Boa] it was shown that the family of languages generated by bidirectional context-free grammars with initial context-free language – thus applying the context-free productions in both a productive as well as in a reductive fashion, starting from a context-free initial set – equals the family of recursively enumerable languages. In the unidirectional case we have that the family of languages generated by context-free grammars with initial context-free language equals the family of context-free languages.

However, in the same paper Boasson introduces an interesting type of grammar, involving bidirectional rewriting. He defines a subfamily of the family of context-free languages, the family of the so-called nonterminal separating or NTS languages, as follows. Let G be a context-free grammar (V, Σ, P, S). Denote by $\Rightarrow^*$ the (usual) derivation relation on V^* defined by P. We construct a Thue system on V induced by P by taking the set of rewriting rules equal to P. For each set of nonterminals $M \subseteq V - \Sigma$ the following sets are defined.

- Denote the set of words over Σ derivable from M by G as

$$L(G,M) = \{w \in \Sigma^* \mid \exists A \in M \,.\, A \Rightarrow_P^* w\}.$$

Note that this set equals the language generated by the context-free grammar with initial set M.

- The set of sentential forms generated by G from M is denoted by

$$\underline{L}(G,M) = \{\omega \in V^* \mid \exists A \in M \,.\, A \Rightarrow_P^* \omega\}.$$

- The set of words over V derivable from M by the Thue system P, i.e., by both productions and corresponding reductions, equals

$$\underline{\underline{LR}}(G,M) = \{\omega \in V^* \mid \exists A \in M \,.\, A \Leftrightarrow_P^* \omega\}.$$

If M equals a singleton set $\{A\}$, then we write $L(G,A)$, $\underline{L}(G,A)$, and $\underline{\underline{LR}}(G,A)$, respectively.

A context-free grammar is called *nonterminal separating* (or NTS) if for each A in $V-\Sigma$, we have $\underline{\underline{LR}}(G,A) = \underline{L}(G,A)$. A context-free language L_0 is NTS if there exists an NTS grammar $G = (V, \Sigma, P, S)$ and a set M with $M \subseteq V-\Sigma$, such that $L_0 = L(G,M)$. In that case we write $G = (V, \Sigma, P, M)$ rather than $G = (V, \Sigma, P, S)$. NTS languages are congruential and acceptable by a deterministic pushdown automaton [BoaSén]. A language L_0 over Σ is congruential if L_0 is the union of congruence classes generated by some congruence over Σ with respect to concatenation. In fact, in an NTS grammar $G = (V, \Sigma, P, M)$ for each $A \in V-\Sigma$ the language $L(G,A)$ is a congruence class induced by a finitely generated congruence over Σ, which is induced in its turn by G; cf. [BoaSén] for details. An NTS grammar G has the property that for each two nonterminals A and B, either $L(G,A) \cap L(G,B) = \emptyset$ or $L(G,A) = L(G,B)$ holds. This latter property, the so-called Disjoint Syntactic Category property (DSC), is also used in texts on (parallel) parsing [Lan].

Example 2.3.1. The language $\{a^n b^n \mid n \geq 0\}$ is an NTS language. This can be shown as follows. Consider the context-free grammar G equal to $(\{S,a,b\},\{a,b\},P,S)$, where P equals $\{S \to aSb,\ S \to ab\}$. Then we have $L(G) = L(G,S) = \{a^n b^n \mid n \geq 0\}$. Observe that

$$\underline{L}(G,S) = L(G,S) \cup \{a^n S b^n \mid n \geq 0\}.$$

Each rule in $P \cup \overline{P}$ is applicable to some ω in $\underline{L}(G,S)$. In each case the resulting string is in $\underline{L}(G,S)$. Thus $\underline{\underline{LR}}(G,S) = \underline{L}(G,S)$. □

It is easy to show that the language $L_1 = \{a^n b^n \mid n \geq 1\} \cup \{a^n b^{2n} \mid n \geq 1\}$ can be generated by a context-free grammar that possesses the DSC property. But L_1 is not an NTS language [BoaSén].

From the DSC-property and the fact that NTS languages are deterministic context-free languages one would expect some nice parsing properties and efficient recognition algorithms based on NTS grammars. However, the regular language $\{a\}^+$ can be generated by the NTS grammar

$(\{A,a\},\{a\},\{A \to AA,\ A \to a\},A)$ which hardly possesses any efficient or elegant parsing properties. For instance, with respect to Earley's algorithm [Ear], sentences of this language will be recognized in time proportional to n^3. We did not succeed in obtaining particular and interesting parsing properties for NTS or DSC grammars. So, the question whether there exists more elegant and efficient parsing strategies for NTS or DSC languages remains open.

To conclude this section we note an interesting application of the theory of NTS languages in [FülVág]. In this paper Fülöp and Vágvölgyi prove that the family of congruential languages coincides with the family of recognizable tree languages. This theorem was originally stated, without a proof, by Kozen [Koz]. For additional papers on NTS languages we refer to [AutBoaSén77, AutBoaSén84, Sén81, Sén85, Sén89, Boo81, Fro].

2.4. Macro Grammars

Apart from bidirectional grammars with an underlying context-free grammar, we can also take other types of grammar as underlying grammar. In this thesis we also consider bidirectional grammars based on (variants of) macro grammars.

Macro grammars have been introduced by Fischer in [Fis68a] as a generalization of context-free grammars. The difference with context-free grammars is that in a macro grammar we associate with each nonterminal a nonnegative number of arguments. We take arguments from the set of terms build up from nonterminals, terminals, and variables. Therefore we can consider a macro grammar as a particular kind of term rewriting system. In order to define macro grammars in a precise way we use the concepts of ranked alphabet, and term over a ranked alphabet.

A *ranked alphabet* Δ is an alphabet of which each symbol is provided with a nonnegative integer, called its *rank*. The ranked alphabet Δ is partitioned into sets Δ_i consisting of those symbols with rank equal to i. Thus $\Delta = \cup\Delta_i$ and if $i \neq j$, then $\Delta_i \cap \Delta_j = \emptyset$. Let PC be a set of punctuation symbols, consisting of the left and right parenthesis and the comma symbol.

Definition 2.4.1. Let Δ be a ranked alphabet. The set $T(\Delta)$ of *terms* over Δ is the smallest set of strings over $\Delta \cup PC$ such that

(a) $\Delta_0 \cup \{\lambda\} \subseteq T(\Delta)$;

(b) if $t_1, t_2 \in T(\Delta)$, then $t_1 t_2 \in T(\Delta)$;

(c) if $A \in \Delta_n$ and $t_1, \ldots, t_n \in T(\Delta)$, then $A(t_1, \ldots, t_n) \in T(\Delta)$. □

We will write A instead of $A()$ if A has rank zero, thus if $A \in \Delta_0$.

Definition 2.4.2. A *macro grammar* G is 5-tuple (Φ,Σ,X,P,S), where Φ is a ranked alphabet of *nonterminals*, Σ is an alphabet of *terminals*, X is a finite set of *variables*, and S is an element of Φ_0, called the *start symbol*. It is understood that each terminal and variable has rank zero and that the sets Φ, Σ and X are mutually disjoint. The set P consists of *productions* which have the form $A(x_1,\ldots,x_n)\rightarrow t$ with $A\in\Phi_n$, the variables $x_1,\ldots,x_n$ are distinct elements of X, and t is an element from $T(\Sigma\cup\{x_1,\ldots,x_n\}\cup\Phi)$. □

We need the following terminology to define several *modes of derivation* for a macro grammar. A string τ is a *subterm* of a term σ if τ is a term and τ is a substring of σ. A subterm τ of σ occurs at *top level* if there exist subterms τ_1 and τ_2 such that $\sigma=\tau_1\tau\tau_2$. A term over $\Sigma\cup\Phi$ that is a string over Σ is an *expanded term*.

Definition 2.4.3. Let $G=(\Phi,\Sigma,X,P,S)$ be a macro grammar and let σ and τ be terms over $\Sigma\cup\Phi$.

Then we write $\sigma\Rightarrow_{\mathrm{OI}}\tau$ if

- there is a nonterminal A from Φ_n and terms $\xi_1,\ldots,\xi_n$ over $\Sigma\cup\Phi$ such that $A(\xi_1,\ldots,\xi_n)$ is a subterm on top level in σ;
- $A(x_1,\ldots,x_n)\rightarrow t$ is a production from P;
- τ is obtained from σ by replacing the designated term $A(\xi_1,\ldots,\xi_n)$ by t'. The term t' is the result of substituting the terms $\xi_1,\ldots,\xi_n$ for $x_1,\ldots,x_n$ in t, respectively.

The relation $\Rightarrow_{\mathrm{OI}}$ on $T(\Sigma\cup\Phi)$ represents the *OI-derivation mode*, ("outside-in") which can be considered as expanding macros by outermost calls first.

Secondly, we write $\sigma\Rightarrow_{\mathrm{IO}}\tau$ if

- there is a nonterminal A from Φ_n and there are expanded terms $\xi_1,\ldots,\xi_n$ over Σ and $A(\xi_1,\ldots,\xi_n)$ is a subterm of σ;
- $A(x_1,\ldots,x_n)\rightarrow t$ is a production from P;
- τ is obtained from σ in the same way as formulated in the definition of $\Rightarrow_{\mathrm{OI}}$.

The relation $\Rightarrow_{\mathrm{IO}}$ on $T(\Sigma\cup\Phi)$ represents the *IO-derivation mode*, ("inside-out") which can be considered as expanding macros by innermost calls first.

The reflexive and transitive closure of $\Rightarrow_{\mathrm{OI}}$ [$\Rightarrow_{\mathrm{IO}}$] is denoted by $\Rightarrow^*_{\mathrm{OI}}$ [$\Rightarrow^*_{\mathrm{IO}}$, respectively]. □

An *OI-macro* [*IO-macro*] *grammar* is a macro grammar provided with the mode of derivation OI [IO, respectively]. In the sequel, m denotes a mode of derivation. The language $L_m(G)$ generated by an m-macro grammar

$G = (\Phi,\Sigma,X,P,S)$ is the set $\{w \in \Sigma^* \mid S \Rightarrow_m^* w\}$. The sets *OI* and *IO* denote the families of languages generated by OI and IO-macro grammars, respectively. It is a well-known fact that *OI* and *IO* are incomparable [Fis68a]. Both families properly include the family of context-free languages and they are properly included in the family of context-sensitive languages.

Example 2.4.4. [Fis68a]. Consider the macro grammar G defined by $G = (\Phi,\Sigma,X,P,S)$, where $\Phi = \{S,A,F,G\}$, $\Sigma = \{0,1,c\}$, $X = \{x\}$, and P consists of the productions

$$\begin{array}{ll} S \to F(A), & G(x) \to x, \\ F(x) \to F(xA), & A \to 0, \\ F(x) \to G(x), & A \to 1. \\ G(x) \to xcG(x), & \end{array}$$

This macro grammar generates under the OI-mode the language of equal length substrings, i.e.,

$$L_{\mathrm{OI}}(G) = \{w_1 c w_2 \ldots c w_n \mid w_i \in \{0,1\}^+, \mid w_i \mid = m,\ 1 \le i \le n,\ n,m \ge 1\}.$$

Under the IO-mode the grammar G generates the language

$$L_{\mathrm{IO}}(G) = \{w(cw)^n \mid n \ge 0,\ w \in \{0,1\}^+\}. \qquad \square$$

A *basic* term over $\Sigma \cup \Phi$ is a term in which no nonterminal appears within an argument list of another nonterminal, i.e., all macros are non-nested. A *linear basic* term is a basic term in which at most one nonterminal occurs. Then a [*linear*] *basic grammar* is a macro grammar in which the right-hand side of each production is a [linear] basic term. As a direct consequence we have that providing linear basic grammars with the OI and IO-mode of derivation results in two equivalent types of grammars, i.e., $L_{\mathrm{OI}}(G) = L_{\mathrm{IO}}(G)$, where G is a linear basic grammar. So we can speak of linear basic languages without specifying the mode of derivation. Furthermore, the family of linear basic languages, denoted by *LB*, is properly included in $OI \cap IO$ [Fis68a], and it equals the family of *EDTOL* (Extended Deterministic Tabled O Lindenmayer) languages [Dow].

According to Fischer [Fis68a], we can assume that each production in a linear basic grammar has a special form.

Definition 2.4.5. A linear basic grammar $G = (\Phi,\Sigma,X,P,S)$ is in *standard linear form* if each production from P has one of the forms

(i) $A(x_1,\ldots,x_n) \to B(w_1,\ldots,w_k)$, or

(ii) $A(x_1,\ldots,x_n) \to w$,

where $w, w_1, \ldots, w_k$ are words over $\Sigma \cup \{x_1, \ldots, x_n\}$. □

Theorem 2.4.6. [Fis68a]. *For each linear basic grammar one can construct effectively an equivalent linear basic grammar in standard linear form.* □

It is also possible to define the NTS property and related concepts such as the DSC property for macro grammars; cf. Appendix A, where also a few characterization results – similar to those for context-free languages – have been established.

3. Regularly Controlled Bidirectional Grammars

In this thesis we introduce new grammar models by restricting bidirectional context-free grammars and macro grammars. These grammar models are obtained by restricting the set of possible derivations of a bidirectional grammar. This has been inspired by the observation that context-free NTS grammars are in general highly ambiguous. See, e.g. the context-free NTS grammar generating the language $\{a\}^+$ mentioned above. In addition, we observe that each context-free grammar is a special kind of macro grammar. Therefore, in case of bidirectional macro grammars we restrict ourselves to underlying macro grammars which are (variants of) linear basic grammars.

In order to decrease the number of derivations, we can modify bidirectional grammars with respect to several aspects. First, we can restrict the set of subwords of a sentential form that can be rewritten by a rule of the grammar. Secondly, we can prescribe in which order rules ought to be applied, starting from an initial sentential form. This means that a control mechanism is applied to the derivation of sentences. In that case there is a difference between whether or not we continue with the next rule designated by the control mechanism in case the previous rule is not applicable. And thirdly, we can exclude certain reductions from the set of rules.

3.1. Control on Grammars and Rewriting Systems

First we discuss the notion of control. The idea of controlling the application of productions (in a unidirectional grammar) is well known. In order to refer in an explicit way to productions of a grammar, in many grammar models in which some control mechanism is used, a unique name or *label* is attached to each production.

We can distinguish two main approaches to this subject. In an implicitly controlled grammar the control mechanism is incorporated in the productions of such a controlled grammar. Contrary to this type of grammar we have explicitly controlled grammars, in which by a separate language over labels (or productions) the order of application of the productions is

specified in advance.

A representative of the former type is found in the programmed grammars, introduced and investigated by Rosenkrantz [Ros]. In a *programmed grammar* each (phrase-structure) production is labeled, and with each production there are associated two set of labels S and F. After the application of a production to a sentential form the next production to be applied has to be chosen from S (for Success). If a production cannot be applied, a label of a rule to be tried next for application has to be picked from F (for Failure). In addition, the actual application has to be performed as far as possible to the left-side of the current sentential form (left-most rewriting).

Other typical examples of implicitly controlled grammars are matrix grammars [Abr], state grammars [Kas] and ordered grammars [Fri]. Also indexed grammars [Aho] can be reckoned to this kind of grammar.

However, more abundant in the literature are explicitly controlled grammars. This may perhaps be explained by the more general approach in explicitly controlled grammars. An implicitly controlled grammar induces a subset of words over the set of productions. Due to the specific character of the particular control mechanism, one can only obtain these subsets that belong to restricted families of languages. In addition, the motivation for a particular control mechanism is often rather poor [Kha74a]. In case of explicitly controlled grammars one can specify in advance any language family from which the control language ought to be taken. One example is the controlled grammar introduced and studied by Salomaa in [Sal69, Sal70, Sal73].

In Salomaa's controlled grammars we use a phrase-structure grammar $G = (V, \Sigma, P, S)$ and a control language over the set of productions P. Control words are interpreted in two ways. In both interpretations each production π in a control word ought to be applied. In the first sense, the derivation is blocked in case this is not possible. Secondly, in a broader sense, if π is not applicable, then we first check whether π occurs in a previously specified checking set F ($F \subseteq P$). Only if π does, we continue with the next production in the control word, otherwise the derivation is blocked. In addition, there are no restrictions on the choice of which string in a sentential form ought to be rewritten by an applicable production (free application).

Another grammar model in which (explicit) control languages are used, is the one of Ginsburg and Spanier in [GinSpa]. In this thesis we start from this grammatical model.

In the model of Ginsburg and Spanier a controlled phrase-structure grammar consists of a phrase-structure grammar $G = (V, \Sigma, P, S)$ and a control language C. The control acts on the left derivations induced by G. The relation $\Rightarrow_{G,L}$ on $V^* \times V^*$ is defined by

$x \Rightarrow_{G,L} y$ if $x \Rightarrow_G y$ and there are words t,u,v,w in V^* such that $x = uvw$, $y = utw$, and $u \in \Sigma^*$.

The statement $x \Rightarrow_{G,L} y$ can be paraphrased as "The word y is *left derived* from the word x in a single step by the grammar G". The subscript G can be omitted if G is known from the context. The transitive and reflexive closure of $\Rightarrow_L$ is denoted by $\Rightarrow_L^*$. The language of all words over Σ^* which can be obtained by a left derivation induced by G is defined by

$$L_{left}(G) = \{w \in \Sigma^* \mid S \Rightarrow_L^* w\}.$$

It is well known that for each phrase-structure grammar G, $L_{left}(G)$ is a context-free language [Mat].

To obtain controlled left derivations, we consider for each $\pi \in P$ the relation $\Rightarrow_L^\pi$ on $V^* \times V^*$, defined by

$x \Rightarrow_L^\pi y$ if the single left derivation step $x \Rightarrow_L y$ is performed by the production π.

Let C be a control language – i.e., $C \subseteq P^*$ – and c be a control word in C with $c = \pi_1 \ldots \pi_n$ for some n ($n \geq 0$). Then the relation $\Rightarrow_L^c$ on $V^* \times V^*$ is defined by

(i) if c equals the empty word, then $x \Rightarrow_L^c x$, for each $x \in V^*$,

(ii) otherwise, $x \Rightarrow_L^c y$ if and only if there exists words $\omega_i \in V^*$ $(1 \leq i \leq n-1)$, with

$$x \Rightarrow_L^{\pi_1} \omega_1 \Rightarrow_L^{\pi_2} \omega_2 \ldots \Rightarrow_L^{\pi_{n-1}} \omega_{n-1} \Rightarrow_L^{\pi_n} y.$$

If C is taken from a language family K, then the pair (G,C) is called a K-controlled phrase-structure grammar. The language generated by (G,C) is defined by

$$L_{left}(G,C) = \{w \in \Sigma^* \mid \exists\, c \in C \,.\, S \Rightarrow_L^c w\}.$$

Example 3.1.1. Consider the phrase-structure grammar $G = (V, \Sigma, P, S)$, where $V = \{S,D,\#,a\}$, $\Sigma = \{\#,a\}$, and P is defined by

$$\begin{aligned} \pi_0 &= S \to \#S\#, & \pi_3 &= aD \to Daa, \\ \pi_1 &= S \to aSa, & \pi_4 &= \#D \to \#, \\ \pi_2 &= Sa \to DS, & \pi_5 &= S\# \to \#. \end{aligned}$$

We define the regular control language C over P by

$$C = \{\pi_0 \pi_1^n (\pi_2 \pi_3^m \pi_4)^k \pi_5 \mid n \geq 1,\ m,k \geq 0\}.$$

Then we have that $L_{left}(G,C) = \{\#a^{n\,2^n}\# \mid n \geq 1\}$, which can easily be checked. □

In [Kha74a, Kha74b] linear context-free grammars controlled by context-free languages are investigated within this framework. [A *linear context-free grammar* or *LCFG* is a context-free grammar in which the right-hand side of each production contains at most one nonterminal. Furthermore, *LCFL* denotes the family of linear context-free languages]. Khabbaz defines the hierarchy of languages families $\{\mathcal{L}_n \mid n \geq 0\}$ by

(i) $\mathcal{L}_0 = CFL$,

(ii) $\mathcal{L}_{n+1} = CTRL(LCFG, \mathcal{L}_n)$,

where $CTRL(LCFG, \mathcal{L})$ denotes the family of languages generated by $\mathcal{L}$-controlled linear context-free grammars. Then he showed that this hierarchy is a proper one and lies in the family *CSL* of context-sensitive languages, i.e.,

$$CFL = \mathcal{L}_0 \subset \mathcal{L}_1 \subset \mathcal{L}_2 \subset \ldots \subset CSL.$$

In [DusPar] it was shown that $\mathcal{L}_1 = LIND$, the family of linear indexed languages [Aho].

Greibach investigated such controlled phrase-structure grammars (with left-most rewriting) from another point of view [Gre77]. There, for each type of phrase-structure grammar $\mathcal{G}$, $CTRL(\mathcal{G}, . .)$ is considered as an operator on language families; so it maps a language family $\mathcal{L}$ onto another language family $CTRL(\mathcal{G}, \mathcal{L})$. Within this framework, the results of Khabbaz are special instances.

Besides this concept of controlled grammar introduced by Ginsburg and Spanier, similar approaches are possible. We mention the control mechanism on ETOL-systems, cf. e.g. [GinRoz, Nie, Asv77], which can also be applied to context-free grammars as well as to so-called high-level (macro) grammars with outside-in (OI) derivation mode [Vog]. In these grammatical models the application of the productions is in a parallel fashion, rather than the strictly sequential left-most derivation.

Finally, the notion of control applied to rewriting systems has also been investigated in the literature. For example, in [Chot] controlled rewriting systems are defined as a triple (P, Ξ, R), where P is a rewriting system over the alphabet Ξ and R is a regular language over Ξ. Then a derivation relation $\Rightarrow_{P,L}$ is defined by

$x \Rightarrow_{P,L} y$ if $x \Rightarrow_P y$ and there are words t,u,v,w in Ξ^* such that $x = uvw$, $y = utw$, $v \rightarrow t \in P$ and $u \in R$.

Controlled rewriting systems generalize left derivations in a phrase-structure grammar, since a phrase-structure grammar with left derivations can be represented by a controlled rewriting system (P, V, Σ^*).

3.2. Modes of Derivation for RCB Grammars

In our approach, a controlled bidirectional grammar is a pair (G,C) consisting of a (unidirectional) grammar $G = (V, \Sigma, P, S)$ together with a control language C over the productions from P and reductions induced by $\bar{P}$. Together with a controlled bidirectional grammar we define some modes of derivation. In this thesis we mainly investigate controlled bidirectional grammars provided with a regular control language.

For a context-free grammar $G = (V, \Sigma, P, S)$, let $\bar{P}$ be the set of reductions induced by P, i.e.,

$$\bar{P} = \{\bar{\pi} \mid \pi \in P\},$$

where for each π equal to $A \to \alpha$ the rule $\bar{\pi}$ is defined by $\alpha \to A$.

Definition 3.2.1. A *regularly controlled bidirectional grammar* or *RCB grammer* (G,C) consists of

- a context-free grammar $G = (V, \Sigma, P, S)$, called the *underlying* context-free grammar of (G,C), and
- a regular language C over $P \cup \bar{P}$. The language C is called the *control language* of (G,C).

Recall that a member of $P \cup \bar{P}$ is called a *rule* of (G,C). □

The modes of derivation under consideration are constructed from three submodes, each of which has two possible instances. With each mode of derivation m we have a corresponding derivation relation $\Rightarrow_m$. Using these derivation relations $\Rightarrow_m$ we define with each RCB grammar (G,C) an RCB/m language $L_m(G,C)$. Then a language $L_m(G,C)$ is said to be generated by an RCB grammar (G,C) under mode m, or – in other words – by an RCB/m grammar.

The first submode is application from the right. This submode restricts the way of selecting the substring in a sentential form which ought to be rewritten by the current rule in the control word. With respect to this submode we distinguish two mode instances, RS and RA.

Definition 3.2.2.

- Let α be a string in V^* and r a rule in $P \cup \bar{P}$. In the *right-most string* or *RS-mode* the right-most occurrence of the left-hand side of r is selected as the string that ought to be rewritten, under the condition that the string to the right of this occurrence is in Σ^*.
- Let α be a string in V^* and r a rule in $P \cup \bar{P}$. In the *right-most applicable* or *RA-mode* the right-most occurrence of the left-hand side of r in α is selected as the substring that has to be rewritten. □

Note that in Chapter II, III, and IV two slightly different submodes are defined. However, the results obtained are similar; cf. Chapter IV Section 5 for a discussion of the differences.

As an example, in the string Baa, the rule $a \to A$ can only rewrite the right-most a under RS-mode, viz., $Baa \Rightarrow_{RS}^{a \to A} BaA$. This also holds under the RA-mode. The rule $a \to A$ is not applicable to the string aBA under RS-mode, but it does under RA-mode, viz., $aBA \Rightarrow_{RA}^{a \to A} ABA$. So we have that

$$aBA \Rightarrow_{RA}^{(a \to A)(A \to a)} ABa.$$

Remark that a derivation like $aB \Rightarrow_{m}^{\lambda \to A} aBA$, where m equals RS or RA, is consistent with Definition 3.2.2.

The RA-mode as defined in our bidirectional grammar model is in fact the direct incorporation of right-most rewriting into Thue systems. In [NarOtt] a similar mode of rewriting has been defined for Thue systems. It will be clear that the RS-mode, when restricted to (unidirectional) phrase-structure grammars, is the right-most analogue of left-most rewriting in phrase-structure grammars. The choice for selecting the substring to be rewritten from the right end of the sentential form is of course arbitrary. An approach based on selecting from the left end is also possible and yields similar results. We see that in studying bidirectional grammars together with right-most rewriting and the concept of control, we work in the tradition of [GinSpa].

The second submode is concerned with the continuation of the derivational process in case this process is confronted with a non-applicable rule during the application of a control word to an initial sentential form. We investigate two natural instances.

In the *block mode* (*B-mode*) the derivation is stopped in case the current rule in the control word is not applicable. Then the application of this control word will give no contribution to the set of sentential forms. In the *skip mode* (*S-mode*) we discard the non-applicable rule, and try to apply the next rule in the control word. It follows that in S-mode the application of a control word to an initial string always will result in some sentential form. Note that the B and S-mode coincide in the approach of Salomaa [Sal69, Sal70, Sal73] with a checking set F equal to $\varnothing$ and $V - \Sigma$, respectively. In addition, notice that in the approach of Ginsburg and Spanier the B-mode is used. So, our approach is a combination of the approaches in [GinSpa] and [Sal69, Sal70, Sal73].

The third submode arises from the following consideration. In the set of productions P of a context-free grammar $G = (V, \Sigma, P, S)$ we distinguish

productions of the form $A \to \tau$, with $\tau \in \Sigma^*$, from productions of the form $A \to \sigma$, with $\sigma \in V^* - \Sigma^*$. These productions are called terminal and nonterminal productions, respectively. Consequently, in the set of rules $P \cup \bar{P}$ of an RCB grammar $G = (V, \Sigma, P, S)$, reductions associated with terminal productions are called terminal reductions.

Obviously, terminal reductions – considered as independently introduced rewriting rules, i.e., independent of the associated terminal productions – do not fit in the concept of phrase-structure grammar; cf. Definition 2.1.2. So the strict distinction between terminals and nonterminals, as it is expressed in the restrictive form of phrase-structure grammar productions, is lost. As a consequence, we obtain a Thue system to which a control mechanism is applied, and in which we only distinguish a special – terminal – alphabet. We remark that in such a controlled Thue system the adjective "controlled" ought to be distinguished from "controlled" as it is used in the controlled rewriting systems defined by Chottin [Chot].

Now, the third submode consists of allowing only certain types of reductions from the set of rules. It is likely that this will influence the generating power of (controlled) bidirectional grammars. We study two submode instances, motivated by the observations on terminal reductions mentioned above.

First, in the *general mode*, abbreviated by *g-mode*, each reduction is allowed, i.e. the control language is included in $(P \cup \bar{P})^*$. Secondly, in the *fair mode*, abbreviated by *f-mode*, we only allow nonterminal reductions. We call this submode instance fair for it respects the distinction between terminals and nonterminals. Then we are dealing with controlled phrase-structure grammars of a special kind. By fair reductions we mean reductions of the form $\sigma \to A$, $\sigma \in V^* - \Sigma^*$. The set of the corresponding fair (or nonterminal) productions is defined by $P_f = \{A \to \sigma \in P \mid \sigma \in V^* - \Sigma^*\}$. Then we can assume without loss of generality that the (regular) control language of an RCB grammar provided with the fair mode is included in the set $(P \cup \bar{P}_f)^*$; cf. Section 2 of Chapter II.

Now we combine instances of submodes to form composite modes, or modes for short. Each submode has two instances so that we can form eight modes. These are RS/B/f, RS/B/g, RS/S/f, RS/S/g, RA/B/f, RA/B/g, RA/S/f, and finally RA/S/g. Furthermore, we use the following convention. If we do not specify one or more submode instances, then we assume that in each position of "../../.." both of the corresponding possible instances are involved. For example, "RS/f-mode" means "RS/B/f and RS/S/f-mode".

With each (composite) mode m and each rule r in $P \cup \bar{P}$ we define a derivation relation.

Definition 3.2.3. Let m be the mode $\alpha/\beta/\gamma$ and let r be a rule. Then the derivation relation $\Rightarrow_m^r$ over $V^* \times V^*$ is defined by

$$x \Rightarrow_{\alpha/\beta/\gamma}^{r} y$$

if either r is applicable to the string x with respect to α and γ and the result is the string y,

or r is not applicable to x with respect to α and γ, and moreover $\beta = S$ and $x = y$. □

The definition of the relation $\Rightarrow_m^c$, where $c \in (P \cup \bar{P})^*$, can be derived in a straightforward way from the definition of $\Rightarrow_m^r$; cf. p. 16. Then the language generated by the RCB/m grammar (G,C) – i.e., the RCB grammar (G,C) under mode m – is defined by

$$L_m(G,C) = \{w \in \Sigma^* \mid \exists\, c \in C \,.\, S \Rightarrow_m^c w\}.$$

To show the differences between the various modes we present the following example of an RCB grammar (G,C) that yields for each mode a different language.

Example 3.2.4. Consider the RCB grammar (G,C), where $G = (V, \Sigma, P, S)$, with $V = \{S,A,B,D,E,a,b,d,e\}$, $\Sigma = \{a,b,d,e\}$ and P consists of

$$\begin{aligned} \pi_1 &= S \to AaBe, & \pi_5 &= D \to d, \\ \pi_2 &= A \to a, & \pi_6 &= D \to aB, \\ \pi_3 &= A \to b, & \pi_7 &= E \to e, \\ \pi_4 &= B \to b, & \pi_8 &= E \to d. \end{aligned}$$

Define the control language C by

$$C = \{\pi_1\}\{\pi_2, \bar{\pi}_7\}\{\pi_8, \bar{\pi}_6\}\{\bar{\pi}_2, \pi_4, \pi_5\}\{\pi_3, \lambda\}.$$

We observe that C consists of 24 $(= 2 \cdot 2 \cdot 3 \cdot 2)$ control words. This control language C derives, when applied to G, for each mode m a different language. All these languages are listed below, where we only show those derivations that yield terminal strings. In addition, we assume that π_1 has already been applied to S.

- $L_{RS/B/f}(G,C) = \varnothing$.

  In each control word in C, the occurrence of π_2 or $\bar{\pi}_7$ causes blocking.

- $L_{RS/B/g}(G,C) = \{babd\}$.

  Now $\bar{\pi}_7$ is allowed, and we have the derivation

  $$AaBe \Rightarrow^{\bar{\pi}_7} AaBE \Rightarrow^{\pi_8} AaBd \Rightarrow^{\pi_4} Aabd \Rightarrow^{\pi_3} babd.$$

- $L_{RS/S/f}(G,C) = \{bde, babe\}$.

Non-applicable rules can be skipped. This gives us the following derivations.

$$AaBe \Rightarrow^{\{\bar{\pi}_7, \pi_2\}} AaBe \Rightarrow^{\pi_8} AaBe \Rightarrow^{\pi_4} Aabe \Rightarrow^{\pi_3} babe,$$

and

$$AaBe \Rightarrow^{\{\bar{\pi}_7, \pi_2\}} AaBe \Rightarrow^{\bar{\pi}_6} ADe \Rightarrow^{\pi_5} Ade \Rightarrow^{\pi_3} bde.$$

- $L_{RS/S/g}(G,C) = \{babd, bde, babe\}$.

By coincidence, we have $L_{RS/S/g}(G,C) = L_{RS/B/g}(G,C) \cup L_{RS/S/f}(G,C)$.

- $L_{RA/B/f}(G,C) = \{ade\}$.

Now the occurrence of π_2 in each control word from C is applicable, and we obtain

$$AaBe \Rightarrow^{\pi_2} aaBe \Rightarrow^{\bar{\pi}_6} aDe \Rightarrow^{\pi_5} ade.$$

- $L_{RA/B/g}(G,C) = \{babd, ade\}$.

In addition to all words from $L_{RA/B/f}(G,C)$ we have under this mode the derivation

$$AaBe \Rightarrow^{\bar{\pi}_7} AaBE \Rightarrow^{\pi_8} AaBd \Rightarrow^{\pi_4} Aabd \Rightarrow^{\pi_3} babd.$$

- $L_{RA/S/f}(G,C) = \{ade, bde, aabe, babe\}$.

Skipping rules gives the following additional derivations, when compared with $L_{RA/B/f}(G,C)$.

$$AaBe \Rightarrow^{\bar{\pi}_7} AaBe \Rightarrow^{\pi_8} AaBe \Rightarrow^{\pi_4} Aabe \Rightarrow^{\pi_3} babe,$$

$$AaBe \Rightarrow^{\bar{\pi}_7} AaBe \Rightarrow^{\bar{\pi}_6} ADe \Rightarrow^{\pi_5} Ade \Rightarrow^{\pi_3} bde,$$

$$AaBe \Rightarrow^{\pi_2} aaBe \Rightarrow^{\pi_8} aaBe \Rightarrow^{\pi_4} aabe.$$

- $L_{RA/S/g}(G,C) = \{bade, aabe, babd\}$.

Apart from the derivations of each word from $L_{RA/B/g}(G,C)$ we have in addition

$$AaBe \Rightarrow^{\pi_2} aaBe \Rightarrow^{\pi_8} aaBe \Rightarrow^{\pi_4} aabe.$$ □

3.3. RCB Extended Linear Basic Grammars

In case we replace the underlying context-free grammar of an RCB grammar by a macro grammar, we obtain a regularly controlled bidirectional term rewriting system. In this thesis we will concentrate on linear basic

grammars as the underlying grammar. This is due to the fact that context-free grammars are a special kind of macro grammars, and that for some modes m the family of RCB/m languages is equal to the family of recursively enumerable languages. Recall that *LB* is incomparable with *CFL* [EhrRoz]; so taking linear basic grammars as underlying grammars may result in new, interesting language families.

As a generalization of linear basic grammars "extended linear basic grammars" (or "elb grammars") have been introduced. Starting from [Dow] where the words $w, w_1, \ldots, w_k$ in the standard linear form have been replaced by finite languages over $\Sigma \cup \{x_1, \ldots, x_n\}$, via [EngSchVanL] where regular languages have been used instead of finite languages – however, resulting in no additional generating power – the ultimate extension possible in this way of generalizing the concept of linear basic grammar was defined in [AsvEng79] in which *K*-elb grammars have been introduced by replacing each word $w, w_1, \ldots, w_k$ by a language from a given, arbitrary family of languages *K*. The precise definition of this latter grammatical model is as follows.

Definition 3.3.1. Let *K* be a family of languages. An *extended linear basic K grammar* or *K-elb grammar* is a 6-tuple $(\Phi, \Psi, \Sigma, X, P, S)$ where

- Φ is a ranked alphabet of *nonterminals*,
- Ψ is a ranked alphabet of *language names*,
- Σ is a *terminal alphabet*,
- X is a finite set of *variables*,
- $S \in \Phi_0$ is the *start symbol*,
- P is a finite set of *productions*. Each production has one of the following forms.

$$A(x_1, \ldots, x_n) \to B(\psi_1(\vec{x}), \ldots, \psi_k(\vec{x})), \tag{i}$$

where $A \in \Phi_n$ $(n \geq 0)$, $B \in \Phi_k$ $(k \geq 1)$, and $\vec{x}$ is the abbreviation of $x_1, \ldots, x_n$. Thus $\psi_i \in \Psi_n$ $(1 \leq i \leq k)$. If $A = S$, then production (i) is a so-called *initial* production.

$$A(x_1, \ldots, x_n) \to \psi(x_1, \ldots, x_n), \tag{ii}$$

where $A \in \Phi_n - \{S\}$ and $\psi \in \Psi_n$.

$$\psi(x_1, \ldots, x_n) \to L_0, \tag{iii}$$

where $\psi \in \Psi_n$ and $L_0 \subseteq (\Sigma \cup \{x_1, \ldots, x_n\})^*$ is a language in *K*.

Moreover, we require that for each language name ψ from Ψ there is exactly one production of the form (iii) in *P*. □

We apply the same conventions on notation as we did in Definition 2.4.2, i.e., the list $x_1, \ldots, x_n$ consists of distinct elements of X, the sets Φ, Ψ, Σ and X are mutually disjoint, variables and terminals have rank zero, and if A is an element of either Φ_0 or Ψ_0 we write A instead of $A()$.

In Definition 3.3.1 we demand that $k \geq 1$ in a production of the form (i), whereas in the original definition in [AsvEng79] $k \geq 0$ is permitted. However, we obtain no loss of generality; cf. Section V.2.

Notice that each word w, $w_1, \ldots, w_k$ in the standard linear form is replaced by a language name $\psi(x_1, \ldots, x_n)$, $\psi_1(x_1, \ldots, x_n)$, ..., $\psi_k(x_1, \ldots, x_n)$, respectively. These language names constitute a special ranked alphabet. We associate to each language name ψ a unique production of the form $\psi(x_1, \ldots, x_n) \to L_0$, where L_0 is a language from the family K with $L_0 \subseteq (\Sigma \cup \{x_1, \ldots, x_n\})^*$. As in [Asv78, AsvEng79] this approach allows us to make a distinction between OI and IO-derivations in a natural way. In particular, this implies that for many instances of the family K, the generating power of K-elb grammars depends on the mode of derivation.

The relation of K-elb grammars with macro grammars can be obtained in a natural way when we treat a K-elb grammar $G = (\Phi, \Psi, \Sigma, X, P, S)$ as a macro grammar G' with a countable (rather than a finite) number of productions. Viz., let G' be the macro grammar $(\Phi \cup \Psi, \Sigma, X, P', S)$ where the (countable) set P' of productions is determined by G as follows.

(1) Each production in P of the form 3.3.1(i) or 3.3.1(ii) is also in P'.

(2) For each production $\psi(x_1, \ldots, x_n) \to L_0$ of the form 3.3.1(iii) in P, P' contains the (countable number of) productions
$\psi(x_1, \ldots, x_n) \to w$ for each w in L_0.

Now we can provide this countable macro grammar G' with either the OI-mode or with the IO-mode of derivation. In this indirect way one can define a K-elb grammar with the OI or IO-mode of derivation. Viz., an (m,K)-*elb grammar* G is a K-elb grammar provided with the mode m if the corresponding G' is an m-macro grammar with a countable set of productions. The language $L_m(G)$ generated by G is defined by $L_m(G) = L_m(G')$. The set $LB_m(K)$ denotes the family of languages generated by (m,K)-elb grammars.

We illustrate the concepts defined above by the following example. Let *FIN* be the family of finite languages.

Example 3.3.2. Consider the (m,FIN)-elb grammar G defined by $G = (\Phi, \Psi, \Sigma, X, P, S)$, where $\Phi = \{S, A\}$, $\Psi = \{\psi_0, \psi_1, \psi_2, \psi_3, \psi_4\}$, $X = \{x, y\}$, $\Sigma = \{0, 1\}$ and P consists of the productions

$$\pi_0 = S \to A(\psi_0, \psi_1), \qquad \pi_4 = \psi_1 \to \{0, 1\},$$

$$\pi_1 = A(x,y) \to A(\psi_2(x,y),\psi_3(x,y)), \qquad \pi_5 = \psi_2(x,y) \to \{y\},$$
$$\pi_2 = A(x,y) \to \psi_4(x,y), \qquad \pi_6 = \psi_3(x,y) \to \{yx\},$$
$$\pi_3 = \psi_0 \to \{0,1\}, \qquad \pi_7 = \psi_4(x,y) \to \{y\}.$$

One can verify that

(*i*) $L_{\mathrm{IO}}(G) = \cup\{h_{uv}(L_0) \mid u,v \in \Sigma\}$,

where for each $u,v \in \Sigma$ the length-preserving homomorphism $h_{uv} : \Sigma \to \Sigma$ is defined by $h_{uv}(0) = u$ and $h_{uv}(1) = v$. The language L_0 equals

$$\{1, 10, 101, 10110, 10110101, 1011010110110, \ldots\},$$

which is the set of Fibonacci words over the alphabet $\{0,1\}$. These Fibonacci words are given by the sequence $f : \mathbb{N} \to \Sigma^*$ defined by

$$f_0 = 1\,; \quad f_1 = 10\,; \quad f_{n+2} = f_{n+1}f_n \quad \text{for each } n \ \ (n \geq 0).$$

(*ii*) $L_{\mathrm{OI}}(G) = \sigma(L_0) = \cup\{\{0,1\}^{F_n} \mid n \geq 1\}$,

where the length-preserving substitution $\sigma : \Sigma \to 2^{\Sigma}$ is defined by $\sigma(0) = \Sigma$ and $\sigma(1) = \Sigma$. And F_n is the nth Fibonacci number; i.e., $F_0 = 0$, $F_1 = 1$, $F_{n+2} = F_n + F_{n+1}$ for each n $(n \geq 0)$.

In proving (*i*) we first observe that for all $w_1, w_2 \in \Sigma^*$ we have

$$A(w_1,w_2) \Rightarrow_{\mathrm{IO}}^{\pi_1\pi_6\pi_5} A(w_2, w_2w_1) \tag{1}$$

and

$$A(w_1,w_2) \Rightarrow_{\mathrm{IO}}^{\pi_2\pi_7} w_2. \tag{2}$$

These subderivations can be used to prove by induction that

$$A(0,1) \Rightarrow_{\mathrm{IO}}^{d_n} f_n, \ \text{ for all } n \ (n \geq 0), \tag{3}$$

where $d_n = (\pi_1\pi_6\pi_5)^n\pi_2\pi_7$, for all n $(n \geq 0)$.

It is straightforward to show that

$$\{w \in \Sigma^* \mid A(0,1) \Rightarrow_{\mathrm{IO}}^{*} w\} = L_0.$$

Then it follows that for all $u,v \in \Sigma$,

$$\{w \in \Sigma^* \mid A(u,v) \Rightarrow_{\mathrm{IO}}^{*} w\} = h_{uv}(L_0).$$

Property (*ii*) easily follows from the former, considering the linear character of G. □

We define regularly controlled bidirectional (m,K)-extended linear basic grammars, or (m,REG,K)-belb grammars, by a tuple $(G,C,¢)$, where $G = (\Phi,\Psi,\Sigma,X,P,S)$ is an (m,K)-elb grammar. The symbol ¢ does not occur

in Φ, Ψ, Σ or X. We use $\not\Box$ in order to define $\bar{P}$, the set of reductions corresponding to productions in P. We associate with a production π in P of the form $\psi(\vec{x}) \to L_0$ – cf. Definition 3.3.1(iii) – i.e., a (countable) set of productions $\{\psi(\vec{x}) \to t \mid t \in L_0\}$, a corresponding set of reductions, defined by $\{t \to \psi(\gamma_1, \ldots, \gamma_n) \mid t \in L_0\}$, where γ_i is equal to x_i in case x_i occurs in t, and otherwise γ_i equals the symbol $\not\Box$ $(1 \le i \le n)$. Note that $\bar{\psi}(\gamma_1, \ldots, \gamma_n)$ depends on t. This set of reductions is also denoted by $\bar{\pi}$ or even by $L_0 \to \psi(\vec{x})$. Note that $L_0 = \varnothing$ implies that both π and $\bar{\pi}$ are empty.

For example, let π equal $\psi(x,y,z) \to \{axaz, yz\}$. Then π denotes the set $\{\psi(x,y,z) \to axaz, \psi(x,y,z) \to yz\}$, and the corresponding set of reductions is $\{axaz \to \psi(x, \not\Box, z), yz \to \psi(\not\Box, y, z)\}$. Therefore the reduction $\bar{\pi}$ associated with $\bar{\pi}$ is denoted by $\{axaz, yz\} \to \psi(x,y,z)$.

If π is of the form 3.3.1(i) or 3.3.1(ii), then $\bar{\pi}$ is defined by the rewriting rule $B(\psi_1(\vec{x}), \ldots, \psi_k(\vec{x})) \to A(\vec{x})$ and $\psi(\vec{x}) \to A(\vec{x})$, respectively. Then $\bar{P}$ is defined by $\bar{P} = \{\bar{\pi} \mid \pi \in P\}$ as usual.

Finally, C in $(G, C, \not\Box)$ is a regular control language with $C \subseteq (P \cup \bar{P})^*$.

In Chapter V (m, REG, K)-belb grammars are provided with a derivation mode which corresponds to RS/B/f-mode defined for RCB grammars. The resulting grammars are the so-called (r, f, m, REG, K)-belb grammars.

Example 3.3.3. Consider the $(r, f, \mathrm{OI}, REG, \varnothing NE)$-belb grammar (G, C), where $\varnothing NE$ equals the family ONE of all languages consisting of one word, together with the language $\varnothing$, or formally, $\varnothing NE = ONE \cup \{\varnothing\}$. We define the $(\mathrm{OI}, \varnothing NE)$-elb grammar G by $G = (\Phi, \Psi, \{0,1\}, X, P, S)$, with $X = \{x\}$, $\Phi = \{S, A, B, D\}$, $\Psi = \{\psi_1, \psi_2, \psi_3, \psi_4, \psi_5\}$, and P consists of the productions

$$\begin{aligned}
&\pi_0 = S \to A(\psi_1), && \pi_6 = B \to \psi_1,\\
&\pi_1 = A(x) \to A(\psi_2(x)), && \pi_7 = B \to D(\psi_1),\\
&\pi_2 = A(x) \to \psi_3(x), && \pi_8 = D(x) \to \psi_4(x),\\
&\pi_3 = \psi_3(x) \to \{x\}, && \pi_9 = \psi_4(x) \to \{0x\},\\
&\pi_4 = \psi_2(x) \to \{xx\}, && \pi_{10} = D(x) \to \psi_5(x),\\
&\pi_5 = \psi_1 \to \{1\}, && \pi_{11} = \psi_5(x) \to \{x0\}.
\end{aligned}$$

First, a string $A(\psi_2(\ldots(\psi_2(\psi_1))\ldots))$ is generated, with n occurrences of the language name ψ_2 $(n \ge 0)$. Then by the productions π_2 and π_3, followed by a sequence of productions π_4, we obtain a string with two language names ψ_1 at the right end. After each of these ψ_1's has been rewritten into terminal strings in 0^*10^*, we continue to apply productions π_4. This yields again two ψ_1's from which strings in 0^*10^* can be derived. This continues until a completely terminal string is obtained. The total number of language names ψ_1 that show up during this derivation equals 2^n.

Each ψ_1 generates some string in 0^*10^* by the following sequences of rules. Let $c_1 = \pi_5$, $c_2 = \overline{\pi}_6\pi_7\pi_8\pi_9$, and $c_3 = \overline{\pi}_6\pi_7\pi_{10}\pi_{11}$. Then

$$\psi_1 \Rightarrow^{c_1}_{r,f,\mathrm{OI}} 1,\ \psi_1 \Rightarrow^{c_2}_{r,f,\mathrm{OI}} 0\psi_1,\ \text{and}\ \psi_1 \Rightarrow^{c_3}_{r,f,\mathrm{OI}} \psi_1 0.$$

As the control language we take the trivial control language, i.e., $C = (P \cup \overline{P})^*$. We have

$$L_{r,f,\mathrm{OI}}(G,C) = \{w \in \{0,1\}^* \mid \text{the number of 1's in } w \text{ is a power of 2}\},$$

which can easily be checked. Cf. [Fis68a], where it has also been proved that this language can be generated by an OI-macro grammar but not by an IO-macro grammar. □

4. Outline of Chapters II–VII

Instead of the RS-mode the slightly different RN-mode is introduced in Chapter II–IV, and instead of the RA-mode the related RO-mode is studied. However, the results we have obtained in studying the RN and RO-mode rather than the RS and RA-mode hold for the RS and RA-mode as well; cf. Chapter IV, Section 5. Therefore, in this section we present our results of the Chapters II–IV in terms of the RS-mode and the RA-mode.

4.1. Regularly Controlled Bidirectional Grammars

In Chapter II we investigate RCB/*m* grammars, where *m* ranges over various modes of derivation. First, it is shown that for each mode *m*, the family of RCB/*m* languages includes the family of context-free languages (Proposition II.2.4(1)). In addition, if *m* equals the mode RS/B/f, then it is shown that the family of RCB/*m* languages precisely equals the family *CFL* of context-free languages (Proposition II.2.4(2)).

Section II.3 is devoted to establishing closure properties of the families of RCB/*m* languages. First, the closure properties of the family of RCB/RS/B/f languages needs no further investigation, since this family is equal to *CFL*. Concerning the other modes we have the following results.

- The families of RCB languages are closed under (marked) union.
- The family of RCB/RA/B/f languages and the family of RCB/RS/S/f languages are closed under marked concatenation, marked Kleene +, and marked Kleene $*$.
- The family of RCB/RA/S/f languages is closed under marked concatenation.
- The families of RCB/RA/f languages are closed under concatenation.

- The family of RCB/RA/B/f languages is closed under Kleene +, and Kleene $*$.

With respect to the families of RCB/RA languages we have that these families are

- closed under intersection with regular sets,
- closed under context-free substitution,
- closed under inverse homomorphism.

As a corollary, the family of RCB/RA languages is closed under homomorphism. Furthermore, we have the following.

- The family of RCB/RA/B/f languages is closed under substitution.

In establishing these results we use classical proof methods, however, combined with special arrangements in order to handle the presence of reductions, control languages and rewriting from the right (either RA or RS). A typical example is the proof of the closure of the RCB/RA/B/f language family under substitution.

In Section II.4 we establish a normal form theorem for RCB/RS/B/f grammars. For context-free grammars we have the well-known Chomsky Normal Form (CNF). With respect to RCB/RS/B/f grammars we introduce the weak Chomsky Normal Form. A context-free grammar $G = (V, \Sigma, P, S)$ is in weak CNF if each production in P is either of the form $A \to XY$ or $A \to a$, where X and Y are in V and $a \in \Sigma \cup \{\lambda\}$. Recall that in the usual CNF the symbols X and Y ought to be in $V - \Sigma$. Now an RCB/RS/B/f grammar is in weak CNF if its underlying context-free grammar is in weak CNF. We show that each RCB/RS/B/f grammar can be transformed into an RCB/RS/B/f grammar in weak CNF that generates the same language.

The most interesting result of Section II.5 is the existence of a very simple normal form for RCB/f grammars which have a left-linear [or linear] context-free grammar as their underlying grammar. We show that each such LLRCB/f [LRCB/f, respectively] grammar can be transformed into an equivalent LLRCB/f [LRCB/f, respectively] grammar that has the following properties. Its nonterminal alphabet consists of one symbol only, and each control word in the control language ends with a terminal production. Finally, in Section II.6 we generalize our results to controlled bidirectional context-free grammars with control languages from an arbitrary family of languages, rather than from the family of regular languages.

4.2. Time-Bounded Regularly Controlled Bidirectional Grammars.

In Chapter III we continue our investigation of RCB grammars. We observe that in an RCB/*m* grammar (G,C), where $G=(V,\Sigma,P,S)$, there may be sequences of rules c such that for some words ω over V, we have that $\omega \Rightarrow_m^c \omega$. Consequently, if we have a string d over $P \cup \bar{P}$ with $S \Rightarrow_m^d \omega$, and a string e over $P \cup \bar{P}$ such that all strings of the form dc^*e are in C, and $S \Rightarrow_m^{de} w$ with $w \in \Sigma^*$, then it is hard to construct a parser or a recognizer for this RCB/*m* grammar that terminates for each input string. It is unclear whether or not it is possible to transform in an effective way an RCB grammar into an equivalent RCB grammar without such "cycles" in the control language. At this moment no such transformations – which may yield a linear or a polynomial bound on the length of the derivation – are known. The construction of these transformations will probably depend on the mode of derivation under consideration.

In order to get round this unsolved problem, we use the idea of time-bounded grammar to obtain a bound on the derivation length which only depends on the length of the derived sentence by means of some bounding function. Let (G,C) be an RCB/*m* grammar. We first define a partial function $t_{(G,C)}$ from V^* to $\mathbb{N}$ which assigns to a string w the length of the shortest control word that derives w by (G,C), if such a control word exists. Then we define the time function $T_{(G,C)}$ of an RCB grammar as the function from $\mathbb{N}$ to $\mathbb{N}$ which assigns to every $n \in \mathbb{N}$ the maximal value of $t_{(G,C)}(w)$ over all strings w from Σ^n for which there exists a control word c with $S \Rightarrow^c w$. If there is no such string, $T_{(G,C)}(n)$ will be undefined. Furthermore, a function $\phi : \mathbb{N} \to \mathbb{N}$ is referred to as a *bounding function* of (G,C) (or (G,C) is *bounded* by ϕ) if for any natural number n, if $T_{(G,C)}(n)$ is defined then $T_{(G,C)}(n) \leq \phi(n)$.

Time-bounded grammars have originally been introduced in [Gla] to describe the derivational complexity of general phrase-structure grammars. In [Boo71] bounding functions have been used to generate particular language families; thus Chapter III may also be considered as an extension of [Boo71].

In this framework it is now possible to write parsers for ϕ-bounded RCB/*m* grammars (G,C) in the following way (*m* is any mode of derivation). We parse the input string $w \in \Sigma^*$ with $n = |w|$ in a bottom-up way (which is forced by the mode of derivation which will rewrite at the right-hand side of a string), following in reverse the control language C. We increase a counter each time we can apply a rule (i.e., a production or a reduction) according to this control language C. As long as this counter does not exceed $\phi(n)$ we perform the normal parsing actions [AhoUll, Sud], (however, with some

extensions, due to the fact that we have to deal with reductions in the control language as well); otherwise we have to backtrack. Now the fact that (G,C) is bounded by ϕ guarantees that after a long enough but bounded backtracking process, the parser can decide whether or not w is an element of $L_m(G,C)$. For each mode m, the time and space complexity turn out to be exponential and linear in $\phi^2(n)$, respectively.

Section III.2 contains the definition of time-bounded RCB grammars, together with some properties and examples. Here we restrict ourselves to RCB grammars (G,C) in which the underlying grammar G has no λ-productions. These grammars are referred to as λRCB grammars. For each class Φ of bounding functions we define Φ_m as the family of languages generated by λRCB grammars under mode m which are bounded by bounding functions from Φ. For Φ we will mainly take *POLY*, *POLY*(k) and *LIN* which are the families of polynomial functions, of polynomial functions up to degree k and of polynomial functions of degree 1 (linear functions), respectively, all polynomials having coefficients greater than or equal to zero.

Section III.3 is devoted to some closure properties of a few families Φ_m. Depending on the mode of derivation we can show the regular closure properties (union, concatenation, Kleene +), intersection with a regular set, λ-free context-free substitution and substitution. In this section we also establish a weak CNF for bounded λRCB grammars for one particular mode.

In Section III.4 we construct parsers for ϕ-bounded λRCB/m languages. These constructions are performed for a few characteristic modes. The worst-case time complexity of the parser for the RN/B/f-mode, which induces the smallest language family, is already exponential. Finally, Section III.5 contains some concluding remarks.

4.3. Generating Power of RCB/RA grammars.

In Chapter IV we investigate the generating power of RCB/RA grammars. Actually, RCB/RO grammars are investigated; cf. Section IV.5. We show that the (four) families of RCB/RA languages are all equal to the family of recursively enumerable languages. This is obtained by simulating some Turing machine, by an RCB/RA/B/f grammar (G,C). The idea is to simulate each step of the Turing machine computation by a sequence of a single reduction followed immediately by an associated production. Actually, the control language C of the constructed RCB/RA/B/f grammar can be the trivial one, viz., $C = (P \cup \overline{P})^*$.

In Section IV.4 we have tried to incorporate the ideas of Section IV.3 into a possible proof of the conjecture $POLY = \mathbf{NP}$, where *POLY* is as

defined in Chapter III and **NP** is the family of λ-free languages acceptable nondeterministically in polynomial time. This attempt gave no final results, leaving the conjecture unresolved.

Finally, in Section IV.5 we discuss the differences and correspondences between the modes RA and RS defined in Chapter I and the modes RO and RN defined in Chapter II.

4.4. Regularly Controlled Bidirectional Extended Linear Basic Grammars

In Chapter V we introduce bidirectional regularly controlled (m,K)-elb grammars or (m,REG,K)-belb grammars. In Section V.2 we formally define for (m,REG,K)-belb grammars the RS/B/f-mode of derivation. We call the resulting grammars (r,f,m,REG,K)-belb grammars.

Closure properties of the corresponding family $RBLB_{r,f,m}(K)$ are established in Section V.3. It is shown that for both modes OI and IO and under weak assumptions on the family K, the family $RBLB_{r,f,m}(K)$ is closed under the regular operations (union, concatenation, and Kleene +). Furthermore, we establish that if K is a nontrivial family of languages closed under ngsm-mappings, then $RBLB_{r,f,\mathrm{OI}}(K)$ is a full substitution-closed AFL. We also proof – under appropriate conditions on K – that the family $RBLB_{r,f,\mathrm{IO}}(K)$ is closed under intersection with regular languages and deterministic substitution; hence this family is a full QAFL (in the sense of [AsvEng79]) closed under deterministic substitution.

Section V.4 is devoted to determining the language generating capacity of (r,f,m,REG,K)-belb grammars. We show that the language families $RBLB_{r,f,\mathrm{OI}}(\varnothing NE)$ and $RBLB_{r,f,\mathrm{OI}}(OI)$ are equal to the family OI of OI-macro languages, and that the family IO of IO-macro languages is included in the family $RBLB_{r,f,\mathrm{IO}}(\varnothing NE)$. Moreover, we show that the family OI is unequal to the family $RBLB_{r,f,\mathrm{IO}}(\varnothing NE)$.

In Section V.5 we study (m,REG,K)-belb grammars provided with free application of rules, maintaining the restriction of allowing fair reductions only . Then we show that the family of languages generated by these so-called (f,m,REG,K)-belb grammars equals the family of recursively enumerable languages.

4.5. Regularly Controlled Bidirectional Linear Basic Grammars

Linear basic grammars can also serve as underlying grammars in the framework of (regularly) controlled bidirectional grammars. In Chapter VI we define regularly controlled bidirectional linear basic grammars as a tuple

$(G,C,\¢)$, which consists of a linear basic grammar $G=(\Phi,\Sigma,X,P,S)$; cf. Definition 2.4.5, a symbol ¢, and a control language C over $P\cup\overline{P}$, where $\overline{P}$ is defined analogously to the case of (m,REG,K)-belb grammars. I.e., if π equals a production $\psi(x_1,\ldots,x_n)\to t$ in P, where t is either a term of the form $B(w_1,\ldots,w_k)$ or a string w in $(\Sigma\cup X)^*$, then $\overline{\pi}$ is defined by $t\to\psi(x_1,\ldots,x_n)$, where γ_i is equal to x_i in case x_i occurs in t and otherwise γ_i equals ¢ $(1\le i\le n)$. Then $\overline{P}$ is defined by $\overline{P}=\{\overline{\pi}\,|\,\pi\in P\}$. The resulting grammars are called (m,REG)-blb grammars.

We study (m,REG,K)-blb grammars under the RS/B/f-mode, which results in (r,f,m,REG)-blb grammars, or (f,REG)-blb grammars, as will be explained in Section VI.2. We present some interesting examples of (f,REG)-blb grammars and show that each recursively enumerable language L_0 over some alphabet Σ can be obtained by intersecting some (f,REG)-blb language (over an alphabet Γ with $\Gamma\supseteq\Sigma$) with Σ^*.

4.6. Concluding Remarks

In the final Chapter VII we draw some conclusions from the results presented in this thesis in Section VII.1. Some applications are discussed in Section VII.2, and in Section VII.3 we suggest some interesting topics for further research.

4.7. Historical Remarks

Chapter II – VI of this thesis and the appendix have appeared in various media, sometimes in a slightly different form. These chapters differ in their introduction from the original publication. Some other chapters have (slightly) modified sections too, as compared with their first, original form.

Chapter II – Regularly controlled bidirectional grammars – stems from the paper [Hog89a] with the same title published in *Internat. Journal of Computer Math*. In its present form, Sections II.3 and II.4 contain some additions, and Section II.5 has completely been rewritten. The main results of this chapter also appeared in [Hog88a].

Chapter III will be published in nearly the same form in *Internat. Journal of Computer Math*. (to appear). Chapters IV and V originate from two reports of the Department of Computer Science, viz. [AsvHog], which was written together with Peter Asveld, and [Hog89b], respectively. Chapter VI appeared as [Hog90].

Finally, an earlier version of Appendix A has been published in P.R.J. Asveld & A. Nijholt (Eds.): *Essays on Concepts, Formalisms, and Tools* (1987), C.W.I. Tract no. 42, Centre for Mathematics and Computer Science, Amsterdam [Hog88a].

CHAPTER II

Controlled Bidirectional Grammars

1. Introduction

In this chapter we investigate RCB grammars, i.e., context-free grammars the rules of which can be used in a productive and in a reductive fashion, where the application of these rules is controlled by a regular language. We distinguish several modes of derivation for this kind of grammar. The resulting language families (properly) extend the family of context-free languages. In Section 2 various modes of derivation are introduced. Note that the modes used in this chapter differ from the modes defined in Chapter I. However, the obtained results will be similar; cf. Section IV.5 for a discussion of the differences.

In Section 3 we establish some closure properties of the language families defined by RCB grammars. These closure properties consist of the regular ones (union, concatenation, and Kleene +) and closure under homomorphism, inverse homomorphism, intersection with a regular set, and (regular or context-free) substitution. In Theorem 3.6 the most important results are summarized in AFL-terminology as follows. The family of RCB/RO/B/f languages is a full AFL (Abstract Family of Languages) closed under substitution. The family of RCB/RO/S/f languages is a full semi-AFL closed under concatenation. And the family of RCB/RO/g languages is a full semi-AFL.

In Section 4 we introduce the notion of "weak Chomsky Normal Form". This is a variant of the Chomsky Normal Form in which productions of the form $A \to XY$ with X or $Y \in \Sigma$ are allowed. The main result of this section shows that every RCB/RN/B/f language can be generated by an equivalent RCB/RN/B/f grammar in this particular normal form.

Linear and left-linear RCB grammars – abbreviated by LRCB and LLRCB grammars, respectively – are studied in Section 5. Besides some closure properties of the corresponding language families, we also establish a normal form for some modes of derivation. In this normal form an (L)LRCB grammar has a single nonterminal in its underlying grammar only.

Section 6 is mainly devoted to the generalization to arbitrary control languages rather than regular ones. In this way it becomes clear which properties of the (regular and arbitrary) control languages are needed to prove the results of the previous sections.

2. Definitions and Examples

For completeness' sake we recall Definition 3.2.1 of Chapter I.

Definition 2.1. A *regularly controlled bidirectional grammar* or *RCB grammer* (G,C) consists of

- a context-free grammar $G = (V, \Sigma, P, S)$, called the *underlying* context-free grammar of (G,C), and
- a regular language C over $P \cup \bar{P}$. The language C is called the *control language* of (G,C). □

Before defining the language generated by an RCB grammar (G,C), we first consider several *modes of derivation*, i.e., ways in which productions and reductions are applied to a sentential form of the underlying context-free grammar G, according to a word from the control language C. For each mode m, this results in a particular derivation relation $\Rightarrow_m$. Then using these derivation relations, we will associate to each mode m the language $L_m(G,C)$ generated by (G,C) under mode m. Roughly spoken, a terminal word w belongs to $L_m(G,C)$ if and only if it can obtained by means of applying a sequence of productions and reductions from $P \cup \bar{P}$ starting with S, according to some control word in the control language C. In the sequel a member of $P \cup \bar{P}$ will be called a *rule* of (G,C).

First we introduce two ways of selecting the nonterminal symbol from a string α in V^* to which a production π has to be applied, viz.

(1) *RN-mode*: determine the right-most nonterminal symbol of α,

(2) *RO-mode*: determine the right-most occurrence of the left-hand side of π in α.

The choice for determining the selected nonterminal symbol from the right end of α is arbitrary. Clearly, an analogous approach based on the nonterminal symbol selected from the left end is possible too and yields similar results. Let π be a production from P equal to $A \to \sigma$ and let m be either RN or RO. Now if the nonterminal selected by the mode m in a particular sentential form α is equal to the left-hand side A of π, then we say – as usual – that π is *applicable* to α, and we write $app_m(\pi,\alpha,\beta)$ in case β is the result of replacing that selected occurrence of A in α by the right-hand side σ of π. Next we call a reduction ρ, with $\rho = \bar{\pi}$ for some $\pi \in P$, applicable to a string

α if there exists a string β with $app_m(\pi,\beta,\alpha)$, in case we also write $app_m(\rho,\alpha,\beta)$. It will be clear that there is at most one such a string β.

It may happen that in RN-mode the selected nonterminal is not equal to the left-hand side of a production π, and in both modes it may not even occur. With respect to reductions, in RO-mode it is possible that, when applied to a sentential form α, we cannot find a substring σ equal to the left-hand side of the reduction to the right of the right-most occurrence of the nonterminal symbol, if any is present. And in RN-mode, there may be no substring σ of α such that to the right of this σ only terminals occur. In these cases a production or a reduction is not applicable to a sentential form. Then we can follow two different strategies, giving us two additional mode instances independent of the nonterminal-selecting modes. In the *block mode* (*B-mode*) we do not allow to apply any rule to α once we have tried to apply a rule which was not applicable to α. In this mode the derivation relation $\Rightarrow^r_{m/B}$ – where r is a rule, i.e., either a production or a reduction – holds between strings α and β over V if $app_m(r,\alpha,\beta)$ holds. In the *skip mode* (*S-mode*) we still may apply rules to α after we have tried to apply a non-applicable rule with respect to α and m. In this mode the derivation relation $\Rightarrow^r_{m/S}$ holds between α and β, if either $app_m(r,\alpha,\beta)$ or $\neg\, app_m(r,\alpha,\beta) \wedge \alpha = \beta$ holds. Thus in B-mode applying a rule to a string over V may give no result, whereas in S-mode we will always end up with some string from V^*.

Next we define for $x \in (P \cup \bar{P})^*$ the relation $\Rightarrow^x_m$ which is the analogue of $\Rightarrow^*$ in uncontrolled grammars. In this notation m is a combination of different kinds of modes, separated by /'s, for instance RO/S or RN/B. This notational convention will also be applied to other mode instances to be defined in the sequel. Now let $x = r_1 \ldots r_n$ ($n \geq 0$, $r_i \in P \cup \bar{P}$ for $1 \leq i \leq n$). Then $\alpha \Rightarrow^x_m \beta$ holds if there exists strings $\alpha_i \in V^*$ ($1 \leq i \leq n-1$), with

$$\alpha \Rightarrow^{r_1}_m \alpha_1 \Rightarrow^{r_2}_m \alpha_2 \Rightarrow^{r_3}_m \cdots \alpha_{n-1} \Rightarrow^{r_n}_m \beta .$$

With respect to applying a reduction ρ ($\rho \in \bar{P}$) we distinguish the *g-mode* and the *f-mode* as they are introduced in Chapter I. An RCB grammar in f-mode is in fact a special kind of a controlled phrase-structure grammar; cf. the proof of Proposition 2.4.(2). The distinction between f-mode and g-mode is also important when one considers chain rule deletion and when one studies LRCB and LLRCB grammars, i.e., RCB grammars of which the underlying grammar is linear and left-linear, respectively; cf. Section 5.

Thus each RCB grammar will be provided with three different types of modes, each of which may take one out of two values. RN versus RO, B versus S, and g versus f. In the sequel we will combine these mode values in

an obvious fashion which results in notations like “RN/B/f-mode”, and in concepts as “RCB/RO/S/f grammar”. If we do not specify a mode instance in a proposition or example, then we assume that it applies to both instances. For example, “RN/f-mode” means “RN/B/f-mode and RN/S/f-mode”. Thus, in principle we now have 8 different types of grammars. However, not all these combinations of modes are equally important. Some interesting results will be established for certain mode combinations only; cf. Sections 3, 4 and 5. We will return to this matter in Section 6.

For each of the concrete modes of derivation, introduced above, we can now define the language generated by an RCB grammar under that particular mode.

Definition 2.2. Let (G,C) be an RCB grammar with underlying context-free grammar $G = (V, \Sigma, P, S)$ and control language $C \subseteq (P \cup \bar{P})^*$. For each mode m, the language $L_m(G,C)$ generated by (G,C) under mode m is $L_m(G,C) = \{w \in \Sigma^* \mid S \Rightarrow_m^x w, \text{ for some } x \in C\}$. □

In the following example the differences between the four possible combinations of mode instances of two modes are shown. We study the mode instances RO and RN together with the S-mode and B-mode, and we show that these modes are mutually independent.

Example 2.3. Consider the following RCB grammar (G,C) with $G = (\{S,A,B,a,b\},\{a,b\},P,S)$ and P consists of $\pi_1 = S \to AB$, $\pi_2 = A \to a$, $\pi_3 = B \to A$, $\pi_4 = A \to AA$, $\pi_5 = A \to b$. As the control language we take $C = \{c_1, c_2\}$ with $c_1 = \pi_1\pi_2\pi_3\bar{\pi}_4\pi_5$ and $c_2 = \pi_1\pi_2\pi_3\pi_2$. With every combination of mode instances mentioned above, together with the g-mode, we obtain a different language.

$L_{RN/B/g}(G,C) = \emptyset$. This equality holds because in both control words the application of π_2 causes blocking.

$L_{RN/S/g}(G,C) = \{b\}$. Now π_2 is skipped, so we have the derivations $S \Rightarrow_{RN/S/g}^{c_1} b$ and $S \Rightarrow_{RN/S/g}^{c_2} Aa$.

$L_{RO/B/g}(G,C) = \{aa\}$. In this setting, π_2 is applicable. But $\bar{\pi}_4$ in c_1 causes blocking, and c_2 gives $S \Rightarrow_{RO/B/g}^{c_2} aa$.

$L_{RO/S/g}(G,C) = \{aa, ab\}$. Now $\bar{\pi}_4$ is skipped in c_1, and so $S \Rightarrow_{RO/S/g}^{c_1} ab$. □

The generating power of RCB grammars turns out to be rather strong. For instance, the family of context-free languages is included in the family of RCB/m languages, independently of the mode m.

Proposition 2.4. (1) *The family of context-free languages is included in the family of regularly controlled bidirectional languages for each mode of derivation.*

(2) *The family of RCB/RN/B/f languages coincides with the family of*

context-free languages.

Proof. (1) Let $G = (V, \Sigma, P, S)$ be a context-free grammar. Then $L(G) = L_m(G, C)$ for each mode m, where (G, C) is the RCB grammar with $C = P^*$.

(2) Because of (1) we only ought to prove the inclusion from left to right. In [GinSpa] the family of languages $L_C(G)$ generated by phrase-structure grammars G and control languages C has been investigated. In our notation the mode of derivation used in [GinSpa] reads LN/B where LN abbreviates left-most nonterminal (cf. RN-mode), or even, LN/B/f since in [GinSpa] no reductions are considered. For each RCB/RN/B/f grammar (G, C) with $G = (V, \Sigma, P, S)$ we now consider the phrase-structure grammar $G' = (V, \Sigma, P', S)$ where $P' = P \cup \{\alpha \to \beta \mid \beta \to \alpha \in P,\ \alpha \in V^*(V - \Sigma)V^*\}$ and we modify C accordingly into C'. Then $L(G, C) = L_{C'}(G')$ provided in the latter case we take the RN/B/f-mode instead of the LN/B/f-mode. By a "right-most nonterminal" variant of Corollary 1 to Theorem 2.1 from [GinSpa] we obtain that $L_{C'}(G')$, and hence $L(G, C)$, is context-free. □

For some concrete modes, one can easily show that the generating power of RCB grammars increases as compared with the underlying grammar. This fact is illustrated by the following examples.

Example 2.5. Consider the RCB/g grammar (G, C) with $G = (V, \Sigma, P, S)$, $V = \{S\} \cup \Sigma$, $\Sigma = \{a, b, c\}$, and $P = \{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6, \pi_7\}$, the set of productions, defined as $\pi_1 = S \to abc$, $\pi_2 = S \to a$, $\pi_3 = S \to aa$, $\pi_4 = S \to b$, $\pi_5 = S \to bb$, $\pi_6 = S \to c$, and $\pi_7 = S \to cc$. As the control language we take $C = \pi_1(\bar{\pi}_2\pi_3\bar{\pi}_4\pi_5\bar{\pi}_6\pi_7)^*$. Then $L_g(G, C) = \{a^n b^n c^n \mid n \geq 1\}$, as easily can be checked. Note that P contains only terminal productions. □

Example 2.6. [Sal73]. The language in Example 2.5 is also generated by the RCB/RO/f grammar (G_0, C_0) with $G_0 = (V, \Sigma, P, S)$, where $\Sigma = \{a, b, c\}$, $V = \{S, A, B, C\} \cup \Sigma$, and P consists of the productions $\pi_1 = S \to ABC$, $\pi_2 = A \to Aa$, $\pi_3 = B \to Bb$, $\pi_4 = C \to Cc$, $\pi_5 = A \to a$, $\pi_6 = B \to b$, $\pi_7 = C \to c$. The control language C_0 is given by $\pi_1(\pi_2\pi_3\pi_4)^*\pi_5\pi_6\pi_7$. Note that no reductions occur in any derivation of (G_0, C_0). □

Example 2.7. The language $\{a^n b^n c^n \mid n \geq 1\}$ can also be generated by an RCB/RN/S/f grammar (G_1, C_1). Define $G_1 = (V, \Sigma, P, S)$ by $\Sigma = \{a, b, c\}$, and $V = \Sigma \cup \{A, D, S\}$. The set of productions P is $\{\pi_i \mid 0 \leq i \leq 5\}$ with

$$\begin{aligned} &\pi_0 = S \to abc, && \pi_3 = A \to abD, \\ &\pi_1 = S \to abDSc, && \pi_4 = A \to bDb, \\ &\pi_2 = A \to bDa, && \pi_5 = A \to bb. \end{aligned}$$

By the regular expression $\pi_1^*\pi_0(\bar{\pi}_2\pi_3\bar{\pi}_4\pi_5)^*$ we define the control language C_1. It can easily be checked that $L_{RN/S/f}(G_1, C_1)$ equals the desired language. □

3. Closure Properties

In this section we establish some closure properties of the family of languages generated by regularly controlled bidirectional grammars. In the sequel of this section we assume that L_i $(i \geq 1)$ is a language generated by an RCB grammar (G_i, C_i) with $G_i = (V_i, \Sigma_i, P_i, S_i)$. In addition we assume that $N_i \cap N_j = \emptyset$ if $i \neq j$, where $N_i = V_i - \Sigma_i$ for every $i \geq 1$.

If not stated otherwise the results in this section hold for every combination of modes introduced in the previous section. By Proposition 2.4.(2) the family of RCB/RN/B/f languages inherits all closure properties of the context-free languages. Therefore we mainly focus our attention in this section to modes different from RN/B/f.

Proposition 3.1.

- *The families of RCB languages are closed under (marked) union.*
- *The families of RCB/B/f languages and the family of RCB/RN/S/f languages are closed under marked concatenation, marked Kleene +, and marked Kleene $*$.*
- *The family of RCB/RO/S/f languages is closed under marked concatenation.*
- *The families of RCB/RO/f languages are closed under concatenation.*
- *The family of RCB/RO/B/f languages is closed under Kleene +, and Kleene $*$.*

Proof. Union. We construct an RCB grammar (G, C) from (G_1, C_1) and (G_2, C_2) such that $L(G, C) = L_1 \cup L_2$. Consider the grammar $G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, P, S)$ where $S \notin V_1 \cup V_2$, $P = P_1 \cup P_2 \cup \{\pi_1, \pi_2\}$, and $\pi_i = S \rightarrow S_i$ $(i = 1, 2)$. Define the regular control language C by $C = \{\pi_1\}C_1 \cup \{\pi_2\}C_2$. Then $L(G, C) = L(G_1, C_1) \cup L(G_2, C_2)$.

Marked concatenation. Consider the RCB/f grammar (G, C) for $L_1 \# L_2$ with $\# \notin \Sigma_1 \cup \Sigma_2$ defined as follows. Let G be the context-free grammar (V, Σ, P, S) where $V = V_1 \cup V_2 \cup \{S, \#\}$, $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \{\#\}$, S is a new symbol not occurring in $V_1 \cup V_2$, and $P = P_1 \cup P_2 \cup \{\pi_0\}$ with $\pi_0 = S \rightarrow S_1 \# S_2$. As the regular control language we take $C = \{\pi_0\}C_2C_1$. Then we have $L(G, C) = L(G_1, C_1) \# L(G_2, C_2)$.

Marked Kleene +. Define the RCB/B/f or RCB/RN/S/f grammar (G, C) which generates $(L_1\#)^+$, by $G = (V_1 \cup \{S, \#\}, \Sigma_1 \cup \{\#\}, P, S)$ with $P = P_1 \cup \{\pi_0, \pi_1\}$, $S \notin V_1$, $\# \notin \Sigma_1$, $\pi_0 = S \rightarrow S_1\#$, and $\pi_1 = S \rightarrow SS_1\#$. Take as regular control language $C = (\{\pi_1\}C_1)^* \{\pi_0\}C_1$. Then $L(G, C) = (L_1\#)^+$.

Marked Kleene $*$. $(L_1\#)^*$ is also an RCB/B/f or an RCB/RN/S/f language. This follows from a simple change in the last construction; viz. define an

additional element π_2 of P by $\pi_2 = S \to \lambda$, and take as control language $C = (\{\pi_1\}C_1)^*\{\pi_0\}C_1 \cup \{\pi_2\}$.

The corresponding "unmarked" results for families of RCB/RO/f and RCB/RO/B/f languages are obtained in each case by considering # to be a nonterminal instead of a terminal symbol. In addition, P is extended with the production $\pi_\# = \# \to \lambda$. Finally, the control languages are concatenated (to the right) with $\pi_\#$, $\{\pi_\#\}^+$ and $\{\pi_\#\}^*$, respectively. □

The well-known proof to show closure under concatenation does not work for RCB grammars. Viz. consider the RCB grammars (G_1,C_1) and (G_2,C_2) where $G_1 = (V_1,\Sigma_1,P_1,S_1)$, $G_2 = (V_2,\Sigma_2,P_2,S_2)$, $\Sigma_1 = \{a,b\}$, $V_1 = \{S_1,A,B\} \cup \Sigma_1$, $V_2 = \{S_2\} \cup \Sigma_2$, $\Sigma_2 = \{b\}$. The rules of P_1 and P_2 are

P_1		P_2	
π_{11}	$S_1 \to AA$	π_{21}	$S_2 \to b$
π_{12}	$B \to Ab$		
π_{13}	$A \to a$		
π_{14}	$B \to b$		
π_{15}	$A \to B$		

whereas $C_1 = \{\pi_{11}\overline{\pi}_{12}\pi_{13}\pi_{14}\pi_{15}\pi_{14}, \pi_{11}\pi_{13}\pi_{15}\pi_{12}\pi_{13}\}$ and $C_2 = \{\pi_{21}\}$ are the control languages. To generate $L(G_1,C_1)L(G_2,C_2)$ we can simply take as a candidate the grammar (G,C) with $G = (V, \Sigma, P, S)$, $V = V_1 \cup V_2 \cup \{S\}$, $\Sigma = \Sigma_1 \cup \Sigma_2$, $P = P_1 \cup P_2 \cup \{\pi_0\}$, where $\pi_0 = S \to S_1S_2$ and $C = \{\pi_0\}C_2C_1$. However, we do not reach our aim with this construction. For it is easy to see that $L_S(G_1,C_1) = \{ba,aba\}$, and $L_{RN/S}(G_2,C_2) = L_{RN/S}(G_2,C_2) = \{b\}$, but $L_{RN/S}(G,C) = \{bb,abab\}$.

Analogous counterexamples to show that these closure properties hold for certain modes only can easily be constructed.

Proposition 3.2. *The families of RCB/RO languages are closed under intersection with regular languages.*

Proof. Let $L_1 = L(G_1,C_1)$ and R be a regular language, and let $(Q, \Sigma_R, \delta, q_0, F)$ be a deterministic finite automaton which accepts the reversal of R. We construct an RCB/RO grammar (G,C) with $G = (V, \Sigma, P, S)$ such that $L_1 \cap R = L(G,C)$. The set of nonterminals N will be defined as follows. N contains two new symbols S and Z $(S,Z \notin V_1)$ and all triples of the form (u,A,t) where $u,t \in Q$ and $A \in V_1 \cup \{\lambda\}$. To complete N we add a symbol A_a for every $a \in \Sigma_1 \cup \{\lambda\}$. The set Σ of terminals of G equals $\Sigma_1 \cap \Sigma_R$. In order to define P we use the following notational conventions. For each $x \in V_1^*$, with $x = x_1 \ldots x_m$ $(m \geq 0)$, $x_i \in V_1$ $(1 \leq i \leq m)$, we define

$$\tilde{x} = \{(p_0,x_1,p_1)\ldots(p_{m-1},x_m,p_m) \mid p_i \in Q,\ 0 \leq i \leq m\},$$

$$\tilde{\lambda} = \{(p_0,\lambda,p_1) \mid p_0,p_1 \in Q\},$$

and for every p,q in Q

$$\tilde{x}_p^q = \{(p,x_1,p_1)\dots(p_{m-1},x_m,q) \mid p_i \in Q,\ 1 \le i \le m-1\},$$
$$\tilde{\lambda}_p^q = \{(p,\lambda,q)\}.$$

We denote an element from $\tilde{x}$ by $\tilde{x}(p_0,\dots,p_m)$. Consider for every $\pi = A \to \alpha$ in P_1,

$$P_\pi = \{(p,A,q) \to t \mid p,q \in Q,\ t \in \tilde{\alpha}_p^q\}$$

and for every $a \in \Sigma_1 \cup \{\lambda\}$,

$$P_a = \{(p,a,q) \to a \mid p,q \in Q,\ \delta(q,a) = p\}.$$

Because $P_a = \varnothing$ whenever $a \in \Sigma_1 - \Sigma$, we define $P_\Sigma = \cup\{P_a \mid a \in \Sigma \cup \{\lambda\}\}$. Now we define the set P of productions of G by

$$P = P_0 \cup P_F \cup P_E \cup P_\Sigma \cup \cup\{P_\pi \mid \pi \in P_1\}$$

where

$$P_0 = \{S \to Z(u,S_1,q_0) \mid u \in Q\},$$
$$P_F = \{A_a \to Z(u,a,t) \mid u = \delta(t,a),\ u \in F,\ a \in \Sigma_1 \cup \{\lambda\}\},$$
$$P_E = \{A_a \to a \mid a \in \Sigma \cup \{\lambda\}\}.$$

Consider the finite substitution $\sigma : P_1 \cup \bar{P}_1 \to 2^{(P \cup \bar{P})^*}$ defined by $\sigma(\pi) = P_\pi$ and $\sigma(\bar{\pi}) = \bar{P}_\pi$ for each $\pi \in P_1$. Finally, we define the control language C by $C = P_0\sigma(C_1)\bar{P}_F P_E P_\Sigma^*$.

The fact that (G,C) exactly generates $L_1 \cap R$ is shown as follows. Let $T = \bar{P}_F P_E P_\Sigma^*$ and let $w \in L(G,C)$. Then there exist $\pi_0 \in P_0$, $d \in \sigma(C_1)$ and $t \in T$ such that $S \Rightarrow_{RO}^{\pi_0 dt} w$. Applying π_0 yields that there are $p \in Q$, $d \in \sigma(C_1)$ and $t \in T$, such that $Z(p,S_1,q_0) \Rightarrow_{RO}^{dt} w$. From the definition of $\sigma(C_1)$, this d yields $p,p_1,\dots,p_{m-1}$ in Q such that there exist $t \in T$ and $v \in L_1$ with $Z\tilde{v}(p,p_1,\dots,p_{m-1},q_0) \Rightarrow_{RO}^{t} w$. Following the definitions of $\bar{P}_F$, P_E and P_Σ^*, we see that this implies that $p \in F$, $v = w$ and $w \in L_1 \cap R$. The second part of the proof is obtained by traversing this argument in the opposite direction. □

Proposition 3.3.

(a) *The family of RCB/RO/B/f languages is closed under substitution*.

(b) *The families of RCB/RO languages are closed under context-free substitution*.

Proof. (a) Let $L_1 = L(G_1,C_1)$ be an RCB/RO/B/f language and let σ be an RCB/RO/B/f-substitution $\sigma : \Sigma_1 \to 2^{\Sigma^*}$. Assume that $\Sigma_1 = \{a_1,\dots,a_n\}$.

Then for each $a \in \Sigma_1$, there exists an RCB/RO/B/f grammar (G_a, C_a) with $G_a = (V_a, \Sigma, P_a, S_a)$ such that $L(G_a, C_a) = \sigma(a)$. We assume that for every $a \in \Sigma_1$, $N_1 \cap V_a = \emptyset$ and that $N_{a_i} \cap N_{a_j} = \emptyset$ if $i \neq j$ for every $1 \leq i, j \leq n$. Define alphabets $\Delta = \{S_{a_1}, \ldots, S_{a_n}\}$ and $\Omega = \{Z_{a_1}, \ldots, Z_{a_n}\}$. Furthermore, consider an isomorphism $i : V_1 \to N_1 \cup \Omega$ defined by

$$\begin{aligned} i(A) &= A && \text{for each } A \text{ in } N_1, \\ i(a) &= Z_a && \text{for each } a \text{ in } \Sigma_1. \end{aligned}$$

Let $U = \{A \to \alpha \mid A \in N_1, \alpha \in (N_1 \cup \Omega)^*\}$. Then we introduce a control set $T = \cup\{C_a \mid a \in \Sigma_1\}$ and a homomorphism $h : P_1 \cup \overline{P}_1 \to U \cup \overline{U}$ defined as follows

$$\begin{aligned} h(A \to \alpha) &= A \to i(\alpha), \\ h(\alpha \to A) &= i(\alpha) \to A. \end{aligned}$$

Now we can define the RCB/RO/B/f grammar (G, C) which generates the language $\sigma(L_1)$ by $G = (V, \Sigma, P, S)$ where

– $V = \cup\{V_a \mid a \in \Sigma_1\} \cup N_1 \cup \Delta \cup \Omega \cup \{Z\}$

– $P = \cup\{P_a \mid a \in \Sigma_1\} \cup h(P_1) \cup P_Z \cup \{Z \to \lambda\}$ with
$P_Z = \{Z_a \to Z\, S_a \mid a \in \Sigma_1\}$,

– $S = S_1$

and $C = h(C_1)\, P_Z^* T^* \{Z \to \lambda\}^*$

(b) The construction for the proof of Proposition 3.3(b) is nearly the same as the one for the proof of 3.3(a) except for the following details. The language L_1 is an RCB/RO language and the substitution is a context-free substitution. The grammars (G_a, C_a) for $\sigma(a)$ are RCB/RO grammars with $C_a = P_a^*$. Furthermore, we do not need a nonterminal Z which is therefore omitted. Then we write U as $\{A \to \alpha \mid A \in N_1, \alpha \in (N_1 \cup \Delta)^*\}$ and $P = \cup\{P_a \mid a \in \Sigma_1\} \cup h(P_1)$. Consequently, $\{Z \to \lambda\}$ is left out of P and the isomorphism i is defined as $i : V_1 \to N_1 \cup \Delta$ with $i(A) = A$, for each $A \in N_1$ and $i(a) = S_a$, for each $a \in \Sigma_1$. As the control language C we take $h(C_1)\, T^*$.

In order to substantiate our claim that $\sigma(L_1) = L(G, C)$, we only give an informal sketch of the correctness of the construction from which one may provide a formal proof. We use the nonterminal Z to prevent interaction between neighbor parts in a sentential form. This interaction may occur (in case we omit these Z's) when we apply T^* to a string $S_a S_a$ for instance. Take some $c_1, c_2 \in C_a$ ($C_a \subseteq T$) such that c_1 applied to S_a gives no terminal string, and c_2 applied to S_a yields a terminal string w_2. Now it may happen that after applying c_2 to $S_a S_a$ and then c_1 to $S_a w_2$ we can apply some reduction occurring in c_1 to an intermediate string $x w_2$ which uses some terminal

symbols of w_2. Then it might happen that c_2c_1 applied to S_aS_a will yield a terminal string which is not in $L(G_a,C_a)L(G_a,C_a)$, thus violating $\sigma(aa) = \sigma(a)\sigma(a)$. Note that introducing these Z's in order to avoid these interactions properly works for the RO-mode only. The f-mode is of course necessary to prevent terminal reductions which may be applied at the wrong places in a sentential form derived by (G,C). Analogously, this construction is restricted to the B-mode because the S-mode combined with the RO-mode may lead to similar counterexamples as mentioned above. In that case rules may be applied to the wrong sentential forms although they are separated by Z's.

The correctness argument for the proof of 3.3(b) is easier, since in the derivations according to (G_a,C_a) only productions are used, and the control languages C_a are equal to P_a^* for each a in Σ_1. Together with the assumption that the nonterminal alphabets of the grammars G_a are mutually disjoint it is straightforward to prove that $L(G,C) = \sigma(L(G_1,C_1))$. □

Corollary 3.4. *The families of RCB/RO languages are closed under homomorphism.* □

Proposition 3.5. *The families of RCB/RO languages are closed under inverse homomorphism.*

Proof. It is sufficient to prove that RCB/RO languages are closed under intersection with a regular language, regular substitution and union with a regular language; cf. [Gin] Proposition 3.7.1 and its Corollary. The latter fact follows from the observation that the regular languages form a subset of the RCB languages; cf. Proposition 2.4.(1). The other premisses are proven in Propositions 3.2 and 3.3. □

A family of languages is called *nontrivial* if it contains a language which differs from $\emptyset$ and from $\{\lambda\}$. Recall that a *full semi-Abstract Family of Languages* or *full semi-AFL* (cf. [Gin] for this and the following related concept) is a nontrivial family of languages which is closed under union, homomorphism, inverse homomorphisms and intersection with regular languages. Furthermore, a *full Abstract Family of Languages* or *full AFL* is a full semi-AFL which is also closed under concatenation, and Kleene +.

These concepts allow us to summarize some closure properties in the following form.

Theorem 3.6.

- *The family of RCB/RO/B/f languages is a full AFL closed under substitution*.
- *The family of RCB/RO/S/f languages is a full semi-AFL closed under concatenation.*

- *The families of RCB/RO/g languages are full semi-AFL's.*

Proof. These results follow immediately from Propositions 3.1, 3.2, 3.3, 3.5 and Corollary 3.4. □

We define a λRCB grammar to be an RCB grammar of which the underlying context-free grammar G has no λ-productions, i.e. G is λ-free.

Proposition 3.7.

- *The families of λRCB languages are closed under (marked) union.*
- *The families of λRCB/f languages are closed under marked concatenation.*
- *The families of λRCB/B/f languages and the family of λRCB/RN/S/f languages are closed under marked Kleene +.*
- *The families of λRCB/RO/f languages are closed under concatenation.*
- *The family of λRCB/RO/B/f languages is closed under Kleene +.*
- *The families of λRCB/RO languages are closed under intersection with regular languages, and λ-free context-free substitution.*
- *The family of λRCB/RO/B/f languages is closed under substitution.*

Proof. These statements follow immediately from the constructions used in proving Propositions 3.1, 3.2, 3.3 and 3.5. However, the results concerning closure under concatenation and closure under Kleene + are obtained in a way different from the method used in Proposition 3.1. We consider # to be a nonterminal symbol too, but now P is extended with productions of the form $A_a \to a\#$ and $A_a \to a$ with $a \in \Sigma_1$. I.e., let $\Theta = \{A_a \to a\# \mid a \in \Sigma_1\}$, $\Psi = \{A_a \to a \mid a \in \Sigma_1\}$, where the nonterminals A_a do not occur in $V_1 \cup V_2$ or V_1, respectively. Consequently, V is extended with $\{A_n \mid a \in \Sigma\}$. Finally, the control languages are concatenated (to the right) with $\overline{\Theta}\Psi$ and $\overline{\Theta}^*\Psi^*$, respectively. To prove closure under substitution of the family of λRCB/RO/B/f languages we use this method too in order to replace the production $Z \to \lambda$ used in the proof of Proposition 3.3. □

4. Grammatical Transformations

In this section we study certain transformations on RCB grammars with the purpose to obtain normal forms for RCB grammars. First we introduce the notion of "weak Chomsky Normal Form".

Definition 4.1. A context-free grammar $G = (V, \Sigma, P, S)$ is in *weak Chomsky Normal Form* or in *weak CNF* if each production of P has one of the following forms: $A \to XY$ or $A \to a$ with $A \in N$ ($N = V - \Sigma$), where $X, Y \in V$ and a is in $\Sigma \cup \{\lambda\}$. An RCB grammar (G, C) is in *weak CNF* if its underlying

grammar G is in weak CNF. □

We allow X or Y to be an element of Σ, contrary to the usual Chomsky Normal Form where X and Y ought to be members of N only.

To transform an RCB grammar into a weak CNF RCB grammar it is not sufficient to transform the underlying grammar only, but we also ought to modify the corresponding control language. To obtain a weak Chomsky Normal Form for an RCB grammar (G_0, C_0), we first transform it into an equivalent RCB grammar (G_1, C_1) in which G_1 has no chain rules. It turns out that this transformation works properly for one combination of modes only.

Definition 4.2. Let N be a set of nonterminal symbols. A *chain rule* is a rule $A \to B$ with $A, B \in N$, and $CH(N)$ is the set of all chain rules which can be formed with elements from N. □

Lemma 4.3. *Let (G_0, C_0) be an RCB/RN/B/f grammar. Then there exists an equivalent RCB/RN/B/f grammar (G_1, C_1) such that G_1 possesses no chain rules.*

Proof. The idea of the proof is based on similar arguments in [AsvVanL, Asv80] for parallel rewriting systems. Viz. we construct a nondeterministic generalized sequential machine (or ngsm) $T = (Q, P_I, P_O, \delta, q_0, Q_F)$ such that $C_1 = T(C_0)$ and $G_1 = (V_0, \Sigma_0, P_1, S_0)$, with

$$P_1 = \{A \to \omega \mid A \in N_0, A \to \omega \in P_O\},$$

and P_1 has no chain rules. Because the family of regular languages is closed under ngsm-mappings, C_1 is a regular language too. Cf. Chapter V, Definition 4.2, for a precise description of ngsm's and ngsm-mappings.

Each state of T is an ordered pair (X, Y) where X is equal to the right-most nonterminal which appeared in the sentential form by the last non-chain rule in the derivation from S, or it is equal to S itself. The variable Y contains the nonterminal to which X is rewritten by means of a nonempty consecutive sequence of chain rules. The case $Y = \lambda$ denotes that X is not rewritten by chain rules or that it is rewritten by such rules to X itself. The nondeterministic character of T appears when a nonterminal is rewritten to a terminal string. In that case another nonterminal becomes the right-most nonterminal which T ought to guess nondeterministically. The ngsm T also ought to guess whether or not a reduction which is not a chain rule can be applied.

Before giving the formal description of T we introduce the following notation. Let (G, C) be an RCB grammar, r be a rule of (G, C) and let $X \in N$. By $R(\alpha)$ we denote the right-most nonterminal of α if $\alpha \in V^* - \Sigma^*$ and $R(\alpha) = \lambda$ if $\alpha \in \Sigma^*$. Let $lhs(r)$ and $rhs(r)$ denote the left-hand side and the

right-hand side of r respectively. We write r_X to denote the rule $([X/R(lhs(r))]\,lhs(r)) \to rhs(r)$, where $[X/R(\alpha)]\,\alpha$ denotes the string obtained from α by substituting X for the right-most nonterminal of α. Furthermore, we define the set $RN(r)$ as $\{R(rhs(r))\}$ if $rhs(r) \in V^* - \Sigma^*$ and $RN(r) = N \cup \{\lambda\}$ if $rhs(r) \in \Sigma^*$. Finally, we will use a function $act : Q \to N$ defined by

$act((X,Y)) = X$ if $Y = \lambda$ and
$act((X,Y)) = Y$ otherwise.

Now $act((X,Y)) = R(lhs(r))$ is a necessary condition for r to be applicable, and in most cases also sufficient, except when $r \in \overline{P}_0 - CH(N_0)$.

Formally, the ngsm T is defined as follows.

- The set of states is $Q = \{(X,Y) \mid X,Y \in N_0 \cup \{\lambda\}\}$,
- the input alphabet is $P_I = P_0 \cup \overline{P}_0$,
- the output alphabet equals

$$P_O = P_0 \cup \overline{P}_0 \cup \{r_X \mid r \in P_0 \cup \overline{P}_0,\ X \in N_0\} - CH(N_0),$$

- the initial state is $q_0 = (S_0, \lambda)$,
- the set of final states is $Q_F = \{(\lambda,\lambda)\}$,
- the transition mapping $\delta : Q \times P_I \to 2^{Q \times P_O^*}$ is defined by

$$\begin{aligned}
&\delta((X,Y),r) = \\
&\{((Z,\lambda),r) \mid Y = \lambda,\ Z \in RN(r),\ r \notin CH(N_0),\ R(lhs(r)) = X\} \cup \\
&\cup \{((Z,\lambda),r_X) \mid Y \neq \lambda,\ Z \in RN(r),\ r \notin CH(N_0),\ R(lhs(r)) = Y\} \cup \\
&\cup \{((X,\lambda),\lambda) \mid X = rhs(r),\ r \in CH(N_0),\ lhs(r) = act((X,Y))\} \cup \\
&\cup \{((X,rhs(r)),\lambda) \mid X \neq rhs(r),\ r \in CH(N_0),\ lhs(r) = act((X,Y))\}.
\end{aligned}$$

Note that $Y \neq \lambda$ implies $X \neq \lambda$, and consequently r_X is defined.

The correct behavior of T is easily checked. We will only prove for the B-mode that T behaves correctly when it has to guess. Assume that every rule r in a control string is applicable. If r is wrongly considered to be applicable, then – because of the block mode – the output c' of T will block the derivation controlled by c', whenever it tries to apply r. This also holds whenever it tries to apply r_X, which implies $Y \neq \lambda$. We distinguish two cases.

a. If r is a production or $r \in CH(N_0)$, then T produces no output if r is not applicable in the original derivation determined by the control word c, because $act((X,Y)) \neq R(lhs(r))$.

b. In case r is a reduction ρ and $\rho \notin CH(N_0)$, then T ought to guess whether ρ is applicable or not in the original derivation. If ρ is wrongly considered to be applicable, then we have the following situation: ρ is not applicable to a string α with $\alpha = uYv$, $u \in V^*$, $v \in \Sigma^*$, and $S \Rightarrow^b \alpha$, where b is such that there exists an d with $bd \in C$, and finally $R(lhs(\rho)) = Y = act((X,Y)) = R(\alpha)$. The latter holds because T has produced ρ_X. Then $S \Rightarrow^{T(b)} uXv$, and $R(lhs(\rho_X)) = X = R(uXv)$. However, this condition, with X replaced by Y, was apparently not sufficient for ρ to be applicable, so ρ_X is not applicable to uXv.

T also ought to guess the new right-most nonterminal, after a terminal production has been processed by T. Let r be the next rule in the control word c from C_1, i.e., let $c = c_1 r c_2$. Furthermore, let B be equal to $R(w)$, where $S \Rightarrow^{c_1} w$. Note that B may be equal to λ. Suppose the new right-most nonterminal is wrongly guessed to be B' instead of the correct B. Then the new state of T is (B', λ). If $B' = \lambda$, then we have wrongly reached the final state (λ, λ), and so the output c' produces a string not in Σ^*. Now, if $B' \in V - \Sigma$, then we can distinguish two cases.

a. r is a production $\pi = A \to \alpha$. We may suppose $A = B'$. Let $c' = c'_1 r c'_2$ be a produced output of T. Then c' will give no contribution to $L(G,C)$. When applying π to w (with $S \Rightarrow^{c'_1} w$) the derivational process is blocked, because $A = B' \neq B$ and B is the actual right-most nonterminal at that moment.

b. r is a reduction ρ, and in fact it is a fair reduction. Therefore the applicability of ρ depends on B, which is essential. Suppose ρ is inapplicable in the original derivation. In T we suppose ρ to be applicable, so $B' = R(lhs(\rho))$. But then the output c' of T will cause blocking in applying ρ at this place – i.e., after the application of C'_1 in c', if $\rho \notin CH(N_0)$. If $\rho \in CH(N_0)$, then T has constructed a production $B' \to \phi$ with $\phi \notin N_0$ at this place, which will also give blocking. □

For the RN/S/f-mode we are faced with the following difficulties. An eventual ngsm T_S for the RN/S/f-mode will have a transition mapping δ_S with at least the set $\{((X,Y),\lambda) \mid act((X,Y)) \neq R(lhs(r)),\ r \in CH(N_0)\}$ included in $\delta_S((X,Y),r)$. If we extend the mapping δ used at the RN/B/f-mode with this set to obtain δ_S, then we have to deal with the following example. Let $G_0 = (V_0, \Sigma_0, P_0, S)$ with $V_0 = \{A,B,S\} \cup \Sigma_0$, $\Sigma_0 = \{a,b\}$. The production set P_0 consists of $\pi_1 = B \to aS$, $\pi_2 = B \to A$, $\pi_3 = S \to A$, $\pi_4 = A \to bS$, $\pi_5 = S \to a$. With C consisting of the control word $c = \bar{\pi}_1 \pi_2 \pi_3 \pi_4 \pi_5$ we obtain $S \Rightarrow^c ba$. However, in processing c by T_S we have $act((S, \lambda)) = R(lhs(\bar{\pi}_1))$. So we assume $\bar{\pi}_1$ to be applicable, which will lead to $T_S(c) = c' = \bar{\pi}_1 \pi_{4,B} \pi_5$, where $\pi_{4,B} = B \to bS$, and thus $S \Rightarrow^{c'} a$. The

correct output of $T_S(c)$ ought to be $\bar{\pi}_1\pi_{4,S}\pi_5$.

The proof technique of Lemma 4.3 probably works for the RN-mode only, viz. in case of the RO-mode we would need states (in the ngsm T) of the form $((A_1,B_1),\ldots,(A_n,B_n))$ with $N_1=\{A_1,\ldots,A_n\}$ and $B_i\in N_1\cup\{\lambda\}$. If we process a production $\pi=B\to\beta$ with some A_i occurring in β, but $A_i\neq B$, then we ought to remember both (A_i,B_i) – i.e., the current value – as well as the new value (A_i,λ) in case π has been applied right to the right-most A_i. Because of recursion this may lead to an infinite set of states which is not allowed for ngsm's.

Similarly, the restriction to the f-mode is essential in the proof of Lemma 4.3. Since in the RN/g-mode it may happen that a reduction $\alpha\to A_i$, $(\alpha\in\Sigma^*)$ introduces a nonterminal right from the right-most nonterminal. Then we ought to store the current state (A_i,B_i) besides the new state (A_i,λ). Again this may lead to an infinite state set.

By means of Lemma 4.3 we are able to prove the following normal form theorem.

Theorem 4.4. *For every RCB/RN/B/f grammar* (G_1,C_1) *there exists an equivalent RCB/RN/B/f grammar* (G,C) *in weak CNF.*

Proof. By Lemma 4.3 we assume that G_1 has no chain rules. Let $P_1=\{\pi_1,\ldots,\pi_n\}$ be the set of productions of G_1 with $\pi_i=A_i\to B_{i,1}\ldots B_{i,m_i}$. Let P be constructed as follows. Starting with the empty set, adjoin every production of P_1 to P which has a right-hand side with a length smaller than three. Next, for every $\pi_i\in P_1$ with $m_i\geq 3$, construct m_i-1 new productions from this production as follows. We take $\pi_{i,1}=A_i\to B_{i,1}D_{i,1}$, $\pi_{i,2}=D_{i,1}\to B_{i,2}D_{i,2}$, …, $\pi_{i,m_i-1}=D_{i,m_i-2}\to B_{i,m_i-1}B_{i,m_i}$. We assume that the $D_{i,j}$'s are distinct from each other, and that these $D_{i,j}$'s constitute the set D. The productions $\pi_{i,j}$ will be adjoined to P. Now we define a homomorphism $h:P_1\to P^*$ with $h(\pi_i)=\pi_i$ if $m_i\leq 2$ and $h(\pi_i)=\pi_{i,1},\ldots,\pi_{i,m_i}$ if $m_i\geq 3$. Furthermore, for a reduction $\bar{\pi}\in\bar{P}_1$ we define $h(\bar{\pi})=\overline{h(\pi)}$, using $\bar{\pi}\bar{\tau}=\bar{\tau}\bar{\pi}$ for every $\pi,\tau\in P_1$. Finally, we take $C=h(C_1)$ and $G=(V_1\cup D,\Sigma_1,P,S_1)$.

Verifying the correctness of this construction is left to the reader as an easy exercise. □

It is unlikely that the arguments used in establishing Lemma 4.3 and Theorem 4.4 can be modified to obtain an RCB/RN/B/f grammar in the usual Chomsky Normal Form, because of productions of the form $A\to\alpha\beta$ with $\alpha\in\Sigma_1^+$ and $\beta\in V_1^*-\Sigma_1^*$. For then we ought to remember to insert productions $F_a\to a$, $a\in\Sigma_1$ in the new control word after inserting productions which will derive β. Because this may get nested up to any level, an ngsm-mapping is not able to handle this.

It is an interesting question whether we can characterize some of the language families defined by a type of RCB grammar in terms of an other one. The next proposition shows that under some conditions we can construct an equivalent RCB/RO/S grammar in f-mode from an RCB/RO/S grammar in g-mode.

Proposition 4.5. *Let* (G_1,C_1) *be a* λ*RCB/RO/S/g grammar. Then there exists an RCB/RO/S/f grammar* (G,C) *that generates the same language as* (G_1,C_1).

Proof. Let $V = V_1 \cup \{S,Z\}$ be the new alphabet of the grammar (G,C), where S and Z do not occur in (G_1,C_1). Define a mapping

$$\zeta : V_1^+ \to (V_1\{Z\})^* V_1$$

by $\zeta(a) = a$ if $a \in V_1$, and $\zeta(ax) = a\,Z\,\zeta(x)$ if $a \in V_1$ and $x \in V_1^+$. Let P be the new production set of G with

$$P = \{A \to \zeta(\alpha) \mid A \to \alpha \in P_1\} \cup \{S \to Z\,S_1 Z,\ Z \to \lambda\} \cup$$
$$\cup \{A \to Z\,a,\ A \to Z\,A \mid A \to a \in P_1,\ a \in \Sigma_1\}.$$

Next we define a homomorphism $h : P_1 \cup \bar{P}_1 \to (P \cup \bar{P})^*$ as follows

$$\begin{array}{lll} h(A \to \alpha) = A \to \zeta(\alpha) & & \text{if } A \to \alpha \in P_1 \\ h(\alpha \to A) = \zeta(\alpha) \to A & \text{if } |\alpha| > 1 \text{ or} & \\ & \alpha \in V_1 - \Sigma_1 & \text{if } \alpha \to A \in \bar{P}_1 \\ h(\alpha \to A) = (Z\alpha \to A)(A \to ZA) & \text{if } \alpha \in \Sigma_1 & \end{array}$$

Now we define the RCB/RO/S/f grammar by (G,C) with $G = (V, \Sigma_1, P, S)$ and $C = \{S \to Z\,S_1 Z\} h(C_1) (Z \to \lambda)^*$.

That the construction is correct can be seen from the fact that for all strings $\alpha,\beta \in V_1^+$ and rule $r \in P_1 \cup \bar{P}_1$, we have $app_{RO/S/g}(r, \alpha, \beta)$ if and only if $app_{RO/S/f}(h(r), Z\,\zeta(\alpha)\,Z,\ Z\,\zeta(\beta)\,Z)$ holds. This latter formula is defined by $app_m(r_1 r_2, \alpha, \beta)$ if and only if $\exists \gamma (app_m(r_1, \alpha, \gamma) \wedge app_m(r_2, \gamma, \beta))$. □

5. Linear and Left-Linear RCB grammars

This section is devoted to the study of RCB grammars of which the underlying grammar is linear or left-linear. The major part of the results in this section consists of straightforward consequences of propositions established in Sections 3 and 4.

Definition 5.1. If the underlying context-free grammar G of an RCB grammar (G,C) happens to be linear, then we call (G,C) a *linear RCB grammar* or *LRCB grammar*. And (G,C) is a *left-linear RCB grammar* or an *LLRCB grammar* if G is a left-linear grammar. □

All the modes of derivation introduced in Section 2 are applicable to LRCB and to LLRCB grammars as well. However, the grammar types LRCB/RN/B/f and LRCB/RO/B/f, as well as the types LRCB/RN/S/f and LRCB/RO/S/f are strongly equivalent. This equivalence is due to the fact that fair reduction maps linear sentential forms into linear sentential forms, in which case the difference between RN-mode and RO-mode vanishes. The same remark applies to LLRCB grammars.

For LRCB/f and LLRCB/f grammars we can establish a very simple normal form.

Proposition 5.2. *Let* (G,C_0) *be an LRCB/f or an LLRCB/f grammar. Then there exists an equivalent LRCB/f or an LLRCB/f grammar* (G,C)*, respectively, which only possesses one nonterminal symbol, and each control word from C ends with a terminal production.*

Proof. Let (G,C_0) be an LRCB/f or an LLRCB/f grammar. For this type of grammar we can easily construct, using a gsm, a grammar (G,C'_1) where C'_1 is such that for every two consecutive rules r_1 and r_2 in a control word $c \in C'_1$, we have $R(rhs(r_1)) = R(lhs(r_2))$, and that the last rule of each control word in C'_1 is a terminal production. Note that due to the B-mode and S-mode, we actually need two gsm's. (Cf. Lemma 4.3 for the definition of R. In this case R yields the nonterminal of a string $\alpha \in \Sigma^*(V-\Sigma)\Sigma^*$, and $R(\alpha) = \lambda$ if $\alpha \in \Sigma^*$.) If we replace each nonterminal in every rule occurring in G and C'_1 by the start symbol S_0 we obtain a grammar (G,C_1) which possesses one nonterminal symbol. This latter step is now possible because the remaining nonterminals in (G,C'_1) have as their single task to indicate at which position in a sentential form a rule ought to be applied. This can be performed by one unique nonterminal as well. □

The obtained normal form will be called the *1-normal form*.

Proposition 5.3. *The family of* [*left-*] *linear context-free languages is included in the family of* [*left-*] *linear regularly controlled bidirectional languages for each mode of derivation.* □

Clearly, the first construction in the proof of Proposition 3.1 also applies to LRCB grammars. Therefore we have

Corollary 5.4. (1) *The families of LRCB languages are closed under (marked) union.*
(2) *The families of LRCB and LLRCB languages are closed under union with a regular set.*

Proof. (2) It is easy to see that the regular languages form a subset of the LLRCB languages. □

Proposition 5.5. *The families of LRCB/S/f languages are closed under marked concatenation, marked Kleene + and marked Kleene ∗.*

Proof. Let $G = (V, \Sigma, P, S)$ be a context-free grammar. We will use the following homomorphism $h : P \to P \cup \{A \to S_1\alpha \mid A \to \alpha \in P\}$ defined by

$$h(A \to \alpha) = A \to \alpha \qquad \text{if } \alpha \in V^*(V - \Sigma)V^*$$
$$h(A \to \alpha) = A \to S_1\alpha \qquad \text{if } \alpha \in \Sigma^*.$$

In addition, define $h(\bar{\pi}) = \overline{h(\pi)}$ for each $\pi \in P$.

Marked concatenation. Let (G_1, C_1) and (G_2, C_2) be LRCB/S/f grammars generating the languages L_1 and L_2, respectively. Define the LRCB/S/f grammar (G, C), which will generate $L_1 \# L_2$, as follows. G is the linear context-free grammar (V, Σ, P, S_2) with $V = V_1 \cup V_2 \cup \{\#\} \cup \{D_a \mid a \in \Sigma_2\}$, $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \{\#\}$. Let $\pi_\#$ be the production $S_1 \to S_1\#$. Furthermore, let $P_L = \{D_a \to aS_1 \mid a \in \Sigma_2\}$, $P_R = \{D_a \to S_1 a \mid a \in \Sigma_2\}$, and define the production set P by $P = P_1 \cup h(P_2) \cup P_L \cup P_R \cup \{\pi_\#\}$. Now define the control language C by $C = h(C_2)(\bar{P}_L P_R)^* \bar{P}_L \{\pi_\#\} C_1$. Note that the last occurrence of $\bar{P}_L$ in each control word c in C has the effect that if the nonterminal S_1 has not been moved to the ultimate left position of the sentential form by some word in $(\bar{P}_L P_R)^*$, then this $\bar{P}_L$ is applicable, and all rules in $\{\pi_\#\}C_1$ will be skipped. As a result no terminal string is generated. Then $L(G, C) = L_1 \# L_2$.

Marked Kleene +. Let (G_1, C_1) be a LRCB/S/f grammar generating L_1. Assume that (G_1, C_1) is in 1-normal form. Let G be the linear context-free grammar (V, Σ, P, S_1), where $V = V_1 \cup \{\#\} \cup \{D_a \mid a \in \Sigma_1\}$, $\Sigma = \Sigma_1 \cup \{\#\}$ and P is defined as follows. Let $\pi_\# = S_1 \to S_1\#$, and $P_L = \{D_a \to aS_1 \mid a \in \Sigma_1\}$, $P_R = \{D_a \to S_1 a \mid a \in \Sigma_1\}$. Then define the set of productions P by $P_1 \cup h(P_2) \cup P_L \cup P_R \cup \{\pi_\#\}$. With the control language C defined by $C = \{\pi_\#\}(h(C_1)(\bar{P}_L P_R)^* \bar{P}_L \{\pi_\#\})^* C_1$, we obtain $L(G, C) = (L_1\#)^+$.

Marked Kleene ∗. This follows immediately from a small change in the last construction; viz. define $\pi_\lambda = S_1 \to \lambda$ and take $C' = C \cup \{\pi_\lambda\}$. □

Concerning the LLRCB languages we have the following results.

Proposition 5.6.

- *The families of LLRCB languages are closed under (marked) union.*
- *The families of LLRCB/f languages are closed under marked concatenation, marked Kleene + and marked Kleene ∗.*

Proof. Union. Cf. the proof of Proposition 3.1.

To prove the other properties we use the same constructions as in Proposition 5.5. Due to the fact that G is left-linear, we will not need the sets $\{D_a \mid a \in \Sigma_i\}$ $(i = 1, 2)$, P_L and P_R in (G, C). Therefore these closure

properties do also hold for the families of LLRCB/B/f languages. □

Many of the constructions used in Section 3 fail to work in the LRCB and LLRCB case. Therefore we have less results for these language families. However, the families of LRCB/f languages and of LLRCB/f languages turn out to be closed under reversal.

Proposition 5.7. *The families of LRCB/f languages and of LLRCB/f languages are closed under reversal.*

Proof. Let (G_1, C_1) be an LRCB/f grammar which generates the LRCB/f language L_1. Define a homomorphism h on $P_1 \cup \overline{P}_1$ by

$$h(A \to w) = A \to w^R \qquad \text{for each production } A \to w$$
$$h(w \to A) = w^R \to A \qquad \text{for each reduction } w \to A$$

where R is the reversal operation. When we define $G = (V_1, \Sigma_1, h(P_1), S_1)$ and the regular control language $C = h(C_1)$, we have $L(G, C) = L_1^R$.

Clearly, the same construction also applies to LLRCB/f grammars. However, in this case the resulting underlying grammar G is a right-linear context-free grammar. Using standard methods it is now easy to construct an LLRCB/f grammar generating L_1^R. □

6. Arbitrary Families of Control Languages

In this paper we extended regularly controlled context-free grammars to regularly controlled grammars with context-free rules which may be applied in a productive as well as a reductive fashion. In this approach we distinguished several (combinations of) modes of derivation. Some of these combinations have originally been introduced in the literature, i.e., the RN-mode in [GinSpa] (actually the LN-mode, cf. Proposition 2.4.(2)) and the B-mode and S-mode in [Sal69, Sal70, Sal73] using somewhat different names. The introduction of the RO-mode has been inspired by the proof to establish closure under intersection with a regular set; cf. the proof of Proposition 3.2. A similar observation can be made for the f-mode with respect to closure under substitution; cf. the proof of Proposition 3.3. However, the latter mode has also a justification in itself, for in g-mode some terminals play the part of “pseudo-nonterminals”, i.e., they are in the terminal alphabet of the grammar but they can act as a nonterminal, for example a reduction $a \to A$, which is not a phrase-structure rule; cf. Example 2.5. This phenomenon obscures the distinction between terminal and nonterminal symbols in grammatical models.

The closure properties established in Section 3 are summarized in Table 1. We can make the following observations from Table 1. First, we

Closure properties of RCB languages.								
	RN				RO			
	B		S		B		S	
	f	g	f	g	f	g	f	g
union	+	+	+	+	+	+	+	+
marked union	+	+	+	+	+	+	+	+
concatenation	+				+		+	
marked concatenation	+		+		+		+	
Kleene +	+				+			
marked Kleene +	+		+		+			
Kleene $*$	+				+			
marked Kleene $*$	+		+		+			
homomorphism	+				+	+	+	+
intersection with a regular set	+				+	+	+	+
context-free substitution	+				+	+	+	+
union with a regular set	+	+	+	+	+	+	+	+
inverse homomorphism	+				+	+	+	+
substitution	+				+			
substitution into a regular set	+				+			

Table 1.

ought to remark that a table entry which is empty does not mean a negative result, but a problem not yet solved. Concerning the positive results, we see that the combination of the modes B and f enables us to prove all the closure properties listed in the table. Intuitively, this is because in combination with the RO-mode other mode instances can cause “side effects” such as in case of the mode instances S or g. In addition we have the result of Theorem 4.4, which gives us a useful normal form for RCB/RN/B/f grammars. These facts make the B/f-mode the most promising combination of modes, especially the RN/B/f-mode.

In establishing the closure properties of RCB languages we used some (closure) properties of the family of regular languages (“over the alphabet of productions and reductions”). If we generalize from the family of regular languages we ought to know which of these properties are needed to obtain these closure properties of RCB languages. Let **C** denote an arbitrary family of control languages. Then, for instance, closure under (marked) union is

Closure property of CB languages	Closure properties of **C**
(marked) union	marked union
concatenation	concatenation, left and right-marking
marked concatenation	concatenation, left-marking
Kleene +	concatenation, left-marking, Kleene $*$, Kleene +
marked Kleene +	concatenation, left-marking, Kleene $*$
Kleene $*$	union, concatenation, left-marking, Kleene $*$
marked Kleene $*$	union, concatenation, left-marking
intersection by a regular set	union, concatenation, Kleene $*$, reversal, finite substitution
homomorphism	union, concatenation, Kleene $*$, homomorphism
regular substitution	union, concatenation, Kleene $*$, homomorphism
context-free substitution	union, concatenation, Kleene $*$, homomorphism
substitution	union, concatenation, Kleene $*$, homomorphism
union with a regular set	$P^* \in \mathbf{C}$
inverse homomorphism	union, concatenation, Kleene $*$, reversal, finite substitution, $P^* \in \mathbf{C}$
substitution into a regular set	union, concatenation, Kleene $*$, marked union, left and right-marking

Table 2.

provable if **C** is closed under marked union, as one can see from the proof of Proposition 3.1. In Table 2 results are shown based on the analysis of the proof of each closure property. Because **C** is no longer equal to the family of regular languages, we generalize RCB grammars to Controlled Bidirectional grammars (CB grammars). Besides the properties of **C**, also a specific combination of modes is necessary to establish each closure property for CB languages. These modes are not included in the table, but can be extracted in a direct way from the results in Section 3. We conclude this subject with a final remark about the mode RN/B/f. Since most of the closure properties of the family of RCB/RN/B/f languages heavily depend on **C** being the family of regular control languages, cf. Proposition 2.4.(2), we cannot expect to maintain all the closure properties if we generalize to an arbitrary family **C**

of control languages.

To obtain closure properties for the family of **C**-controlled bidirectional languages we often need closure under left or right-marking. A family of languages Φ is closed under *left-* or *right-marking* if for every language $L_0 \in \Phi$ also $\{\#\}L_0 \in \Phi$ and $L_0\{\#\} \in \Phi$, respectively, where # does not occur in the alphabet of L_0.

Consequently, we can also generalize Theorem 3.6 in the following way.

Theorem 6.1. *Let* **C** *be a family of control languages such that for every alphabet P, we have* $P^* \in \mathbf{C}$.

- *The family of CB/RO/S/g languages is a full semi-AFL if* **C** *is closed under union, concatenation, Kleene* $*$*, reversal and finite substitution.*
- *The family of CB/RO/S/f languages is a full semi-AFL closed under concatenation if* **C** *is closed under union, concatenation, Kleene* $*$*, left and right-marking, reversal and finite substitution.*
- *The family of CB/RO/B/f languages is a full AFL closed under substitution if* **C** *is closed under union, concatenation, Kleene + and* $*$*, left and right-marking, reversal and finite substitution.* □

Similarly, as a generalization of Theorem 4.4 we have the following result.

Theorem 6.2. *Let* **C** *be a family of control languages closed under ngsm-mappings. Then for each CB/RN/B/f grammar* (G_1, C_1) *with* $C_1 \in \mathbf{C}$ *we can obtain an equivalent CB/RN/B/f grammar* (G, C) *in weak Chomsky Normal Form (and* $C \in \mathbf{C}$*).* □

CHAPTER III

Time-Bounded Controlled Bidirectional Grammars

1. Introduction

Due to the occurrence of reductions it is possible to have in an RCB grammar (G,C) nonempty control strings d and c with properties as already mentioned at the beginning of Section I.4.2, viz., there exists a string ω with $S \Rightarrow_m^d \omega$ and $\omega \Rightarrow_m^c \omega$. When c^* occurs as part of the control language C, it is hard to construct parsers that terminate for each input string. Till now, no transformations are known that transform in an effective way an RCB grammar (G,C) into an equivalent RCB grammar (G',C') without this undesirable behavior. In particular, we are therefore unable to establish a linear or even a polynomial bound on the derivation length of an RCB grammar.

The problem sketched above raised our interest in the derivational complexity of RCB grammars. So we use concepts as bounding function and time-bounded grammars in order to describe this complexity. For these time-bounded RCB grammars we are able to design parsing algorithms indeed.

This chapter is organized as follows. In Section 2 we recall the definition of time-bounded RCB grammars, together with some properties and examples. We restrict ourselves to λRCB grammars (G,C), i.e., RCB grammars (G,C) in which the underlying grammar G has no λ-productions.

In Section 3 closure properties of a few families of time-bounded λRCB languages are established. In this section we also prove the weak Chomsky Normal Form (cf. Section II.4) for time-bounded λRCB grammars under the RS/B/f-mode.

Section 4 is devoted to the construction of parsers for ϕ-bounded λRCB/m languages. We perform these constructions for a few characteristic modes. The worst-case time complexity of the parser for the RN/B/f-mode, which induces the smallest language family, is already exponential.

Section 5 contains concluding remarks, and some generalizations to arbitrary families of control languages and to less restricted families of bounding functions.

2. Definitions, Examples and Elementary Properties.

First, we introduce time-bounded RCB grammars of which we give some examples. Then we establish some properties of time-bounded RCB grammars and their languages. For all unexplained notations and concepts from parsing theory used in this chapter, we refer to standard texts like [AhoUll, Har, Sud].

We start with another example of an RCB language to which we will return in Example 2.6 and in the proof of Proposition 2.9.

Example 2.1. The language $\{a^{2^n} \mid n \geq 0\}$, which is not context-free, can be generated by an RCB/RN/S/f grammar (G,C). Take $G = (V, \Sigma, P, S)$ with $V = \{S,A,B,D,E,F,G,H,a\}$, $\Sigma = \{a\}$ and P consists of the following productions.

$\pi_0 = S \to a$, $\pi_1 = S \to aa$, $\pi_2 = S \to aAaa$, $\pi_3 = A \to aA$,
$\pi_4 = B \to aAa$, $\pi_5 = B \to AD$, $\pi_6 = D \to aaE$, $\pi_7 = D \to Ea$,
$\pi_8 = F \to aE$, $\pi_9 = F \to a$, $\pi_{10} = G \to aA$, $\pi_{11} = H \to Aa$,
$\pi_{12} = H \to a$.

The control language C is defined by

$$C = \{\pi_0\} \cup \{\pi_1\} \cup \{\pi_2 \pi_3^* (\overline{\pi}_4 \pi_5 \pi_6 (\overline{\pi}_7 \overline{\pi}_8 \pi_9 \pi_6)^* \overline{\pi}_{10} \overline{\pi}_{11} \pi_{12} \pi_{10})^+\}.$$

The grammar (G,C) works as follows. For $m \geq 2$ it produces a string $a^{m-1}Aaa$ by applying $\pi_2 \pi_3^{m-2}$ to S. Next, $\overline{\pi}_4 \pi_5 \pi_6$ rewrites aAa into $AaaE$. So one a to the left of A is removed and one a to the right of A is doubled. By $(\overline{\pi}_7 \overline{\pi}_8 \pi_9 \pi_6)^*$ the nonterminal E moves to the right, doubling each a it encounters. As a consequence, $a^x A a^y$ with $x \geq 1$ and $y \geq 2$ is rewritten into $a^{x-1} A a^{2y}$. Finally, the sequence $\overline{\pi}_{10} \overline{\pi}_{11} \pi_{12} \pi_{10}$ checks if there are no more occurrences of a to the left of A, in which case a terminal string is produced. Now it will be clear that this string is of the form a^{2^m}, with $m \geq 2$. Together with the productions π_0 and π_1 we obtain the intended language. □

In introducing (time-)bounded RCB grammars we first define the time function $T_{(G,C)}$ of an RCB grammar (G,C). This time function $T_{(G,C)}$ is a (partial) function such that for any $n > 0$ for which $T_{(G,C)}$ is defined, $T_{(G,C)}(n)$ bounds the length of the shortest control words that derive all strings of length equal to n which are generated by (G,C). This is a modified version of the original definition by Gladkii [Gla] for general

phrase-structure grammars which has been investigated by Book [Boo71]. First, we define the (partial) function $t_{(G,C)}$ which assigns to a string w the length of the shortest control word deriving w by (G,C) if such a control word exists. In the sequel we only consider λ*RCB grammars*, i.e., the underlying context-free grammar G has no λ-productions at all; cf. Section II.3.

Definition 2.2. For any λRCB grammar (G,C) and every $w \in \underline{L}(G,C)$, where $\underline{L}(G,C) = \{w \in V^+ \mid \exists\, c \in C \bullet S \Rightarrow^c w\}$, let $t_{(G,C)}(w)$ be the least integer k such that there is a control word $c \in C$ deriving w with $|c| = k$ or, equivalently,

$$t_{(G,C)}(w) = \min\{|c| \mid S \Rightarrow^c w,\ c \in C\} \qquad \square$$

The function $t_{(G,C)}$ is partial recursive function. This is easy to show by modifying a similar proof from [Boo71].

Definition 2.3. For every λRCB grammar (G,C) the *time function* $T_{(G,C)} : \mathbb{N} \to \mathbb{N}$ is the function determined by

$$T_{(G,C)}(n) = \begin{cases} \max\{t_{(G,C)}(w) \mid \exists\, c \bullet S \Rightarrow^c w,\ w \in V^n\} & \text{if } \underline{L}(G,C) \cap \Sigma^n \neq \varnothing \\ \text{undefined} & \text{otherwise.} \end{cases}$$

$\square$

Originally the time function T_G of a phrase-structure grammar G has been introduced to serve as a measure of its derivational complexity, cf. [Gla]. In [Boo71] Book used time functions "to define families of languages based on "bounds" on derivational complexity". In this paper we use time functions in a similar way, viz. to restrict the possible control languages C which can generate some language L_0, when given an underlying context-free grammar G. For some function $\phi : \mathbb{N} \to \mathbb{N}$, context-free grammar G and two control languages C_1, C_2 it is possible to have $L(G,C_1) = L(G,C_2) = L_0$ and $\forall n \bullet T_{(G,C_1)}(n) \leq \phi(n)$ but not $\forall n \bullet T_{(G,C_2)}(n) \leq \phi(n)$. The function ϕ will be called a bounding function.

Definition 2.4. A function ϕ is a *bounding function* if it is a nondecreasing total recursive function with the property that there is a positive integer k such that for all x, $\phi(x) \geq x/k$ and such that for all $x \geq 0$, $\phi(x) \geq 0$. $\square$

Let Φ denote a family of bounding functions. In this paper we will consider mainly the following families of bounding functions: *POLY*, *POLY*(k) with $k \geq 1$ and *LIN* which are the families of polynomial functions, of polynomial functions up to degree k and polynomial functions of degree 1 (linear functions), respectively, all polynomials having coefficients greater than or equal to zero. Note that *POLY*(1) = *LIN*.

For a partial function $F : A \to B$ we write $F(a)\downarrow$ whenever $F(a)$ is defined and $F(a)\uparrow$ otherwise.

Definition 2.5.

(a) A λRCB grammar (G,C) is *bounded* by a function ϕ if for any natural number n, if $T_{(G,C)}(n)\downarrow$ then $T_{(G,C)}(n)\leq\phi(n)$.

(b) A λRCB language L_0 is *bounded* by a function ϕ if there is a λRCB grammar (G,C) generating L_0 which is bounded by ϕ.

The family of ϕ-bounded λRCB/m languages, denoted by $\mathbf{L}_m(\phi)$, consists of those languages for which there is a λRCB/m grammar (G,C) that is bounded by ϕ. For each class Φ of bounding functions, and for each mode m the family of Φ-bounded λRCB/m languages — denoted by Φ_m — equals $\cup\{\mathbf{L}_m(\phi)\,|\,\phi\in\Phi\}$. □

Example 2.6. The grammar (G,C) of Example 2.1 is bounded by $\phi: n\mapsto 5n$. This is shown as follows. For each $n\in\mathbb{N}$ there is at most one string w from $L(G,C)$ with length n. Furthermore, every string $w\in L(G,C)$ has length 2^m, for some $m\in\mathbb{N}$. Since $S\Rightarrow^{\pi_0}a$ and $S\Rightarrow^{\pi_1}aa$, we have $1\leq\phi(1)$, and $1\leq\phi(2)$. We also have the derivation $S\Rightarrow^{\pi_2\pi_3^{m-2}}a^{m-1}Aaa$, $m\geq 2$, with $|\pi_2\pi_3^{m-2}|=m-1$. Let Δ denote the set $\{\bar{\pi}_4\pi_5\pi_6(\bar{\pi}_7\bar{\pi}_8\pi_9\pi_6)^*\bar{\pi}_{10}\bar{\pi}_{11}\pi_{12}\pi_{10}\}$ and let $d\in\Delta$. Then for $y\geq 2$ we have

$$\left.\begin{array}{l} aAa^y\Rightarrow^d a^{2y}, \\ a^{m-1}Aa^y\Rightarrow^d a^{m-2}Aa^{2y},\ \ m>2 \end{array}\right\}\quad \text{with } |d|=4y+7,$$

where $|d|=4y+7$ implies that the sequence $\bar{\pi}_7\bar{\pi}_8\pi_9\pi_6$ has been repeated y times. If we combine these facts we obtain that there exists an $e\in(P\cup\bar{P})^*$ with

$$S\Rightarrow^{\pi_2\pi_3^{m-2}}a^{m-1}Aaa\Rightarrow^e a^{2^m},\ e\in\Delta^*\quad\text{and}\ |e|=\sum_{i=1}^{m-1}(4\cdot 2^i+7),$$

so that there exists a $c\in C$ with $S\Rightarrow^c a^{2^m}$, $|c|\geq m-1+7m-7+4\cdot 2^m-8=4(2^m+2m-4)$, $(m\geq 2)$. Now we have $\forall m\geq 2\,.\,5\cdot 2^m\geq 4(2^m+2m-4)$ which gives us the linear bounding function $\phi: n\mapsto 5n$. A "sharper" bounding function is of course $\psi: 1\mapsto 1,\ 2\mapsto 1,\ n\mapsto 4(n+2\lceil\log n\rceil-4)$, where $n\geq 3$. □

A useful property for ϕ-bounded λRCB languages is the following characterization, of which the proof is straightforward.

Lemma 2.7. *Let (G,C) be a λRCB grammar. Then for each mode m the following statements are equivalent.*

(1) *$L(G,C)$ is bounded by ϕ.*

(2) $\forall w\in L(G,C)\,.\,\exists\, c\in C\,.\,(S\Rightarrow^c w\wedge|c|\leq\phi(|w|))$. □

Let *CFL* [λ*CFL*] denote the family of [λ-free] context-free languages. The following lemma is a simple modification of Proposition II.2.4.(2); the proof is also a straightforward variation of the original proof.

Lemma 2.8. *The family of λRCB/RN/B/f languages coincides with the family λCFL.* □

Concerning the various families of bounding functions Φ discussed above we have the following result, where λRCB/*m* denotes the family of languages generated by λRCB/*m* grammars.

Proposition 2.9.

(a) *For every family Φ of bounding functions, and for all modes m, we have* $\Phi_m \subseteq \lambda RCB/m$.

(b) *For all modes m, we have* $\lambda CFL \subseteq LIN_m \subseteq POLY(k)_m \subseteq POLY_m$.

(c) *For all modes m ≠ RN/B/f, we have* $\lambda CFL \subset LIN_m$.

Proof. (a) is trivial, and for (b) we use for the first inclusion the fact that every λ-free context-free language can be generated by a λRCB/*m* grammar (G,P^*). Without loss of generality we may take *G* in standard 2-form, i.e., all productions have one of the following three forms: $A \to a$, $A \to aB$, $A \to aBC$, with $a \in \Sigma$, where *S* does not occur at the right-hand side of a production. From this the result easily follows. The other inclusions are trivial.

Finally, (c) can be proved by using the language $L_0 = \{a^n b^n c^n \mid n \geq 1\}$ in case of the modes g and RO/f. For these modes simple RCB/*m* grammars have been constructed in Chapter II which generate L_0. These grammars can easily be shown to be linearly bounded λRCB/*m* grammars. For the mode RN/S/f Example 2.6 establishes the result. □

In Example II.2.7 a λRCB/RN/S/f grammar has also been constructed that generates L_0. However, that grammar is bounded by a polynomial of degree two.

Remark. The case of Φ_m versus RCB/*m* leads to the proper inclusion $\Phi_m \subset$ RCB/*m*, which is shown by considering the language {λ} which can be generated by an RCB/*m* grammar with a single production π equal to $S \to \lambda$ and $C = \{\pi\}$. However, by definition λRCB/*m* grammars cannot have λ-rules. Consequently, Φ_m is a λ-free family of languages. □

Corollary 2.10. $\lambda CFL = LIN_{RN/B/f} = POLY(k)_{RN/B/f} = POLY_{RN/B/f}$.

Proof. This follows directly from Proposition 2.9(a) and Lemma 2.8. □

3. Closure Properties and Normal Form.

In this section we investigate the closure properties of some families of time-bounded λRCB languages. In addition a normal form for some grammars will be established. If not stated otherwise the results in this section hold for every combination of modes mentioned in the previous section.

By Corollary 2.10 the family $\Phi_{RN/B/f}$ ($\Phi = LIN$, $POLY(k)$ or $POLY$) shares all closure properties of the λ-free context-free languages. Therefore we restrict our attention to modes different from RN/B/f. Cf. Table 1 in Section 5.

In the sequel we suppose that (G_i, C_i) are λRCB grammars, where $G_i = (V_i, \Sigma_i, P_i, S_i)$, which are bounded by some $\phi_i \in POLY(k)$ ($i = 1,2$). In addition L_i denotes the language generated by (G_i, C_i), i.e., $L_i = L(G_i, C_i)$. Furthermore, N_i equals the set $V_i - \Sigma_i$, i.e., the set of nonterminals of G_i.

Proposition 3.1. *Let Φ be a family of bounding functions equal to LIN, $POLY(k)$ or POLY. Then the following statements hold.*

- *For all modes m, the families Φ_m are closed under union.*
- *The families $\Phi_{B/f}$ and the family $\Phi_{RN/S/f}$ are closed under marked concatenation and marked Kleene +.*
- *The families Φ_f are closed under marked concatenation.*
- *The families $\Phi_{RO/f}$ are closed under concatenation.*
- *The family $\Phi_{RO/B/f}$ is closed under Kleene +.*

Proof. Union. We construct a λRCB grammar (G,C) from (G_1,C_1) and (G_2,C_2) such that $L(G,C) = L_1 \cup L_2$. Consider the grammar $G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, P, S)$ where $S \notin V_1 \cup V_2$, $P = P_1 \cup P_2 \cup \{\pi_1, \pi_2\}$, and $\pi_i = S \to S_i$ ($i = 1,2$). Define the regular control language C by $C = \{\pi_1\}C_1 \cup \{\pi_2\}C_2$. Then $L(G,C) = L(G_1,C_1) \cup L(G_2,C_2)$. To show that (G,C) is a Φ-bounded λRCB grammar we write

$$T_{(G,C)}(n) \leq 1 + \max\{T_{(G_i,C_i)}(n) \mid i = 1,2\}.$$

Now it is clear that for $\Phi = POLY_m(k)$ it holds that there exists a $\phi \in \Phi$ with $T_{(G,C)}(n) \leq \phi(n)$.

Marked concatenation. The proof for this case is left to the reader as an exercise.

Marked Kleene +. Define the λRCB/B/f or λRCB/RN/S/f grammar (G,C) which generates $(L_1\#)^+$, by $G = (V_1 \cup \{S,\#\}, \Sigma_1 \cup \{\#\}, P, S)$ with $P = P_1 \cup \{\pi_0, \pi_1\}$, $S \notin V_1$, $\# \notin \Sigma_1$, $\pi_0 = S \to S_1\#$, and $\pi_1 = S \to SS_1\#$. Take as regular control language $C = (\{\pi_1\}C_1)^*\{\pi_0\}C_1$. Then $L(G,C) = (L_1\#)^+$. We show that (G,C) is a $POLY(k)_m$ grammar (with the proper modes m) as

follows. For $l \geq 1$, $s_i \geq 1$, let

$$n = \sum_{i=1}^{l} s_i.$$

Write $\phi_1 \in POLY(k)$ as

$$\phi_1(n) = \sum_{j=0}^{k} a_j n^j$$

where $a_k > 0$ and $a_j \geq 0$ $(0 \leq j < k)$. Then we have

$$T_{(G,C)}(n) \leq \sum_{i=1}^{l} (1+\phi_1(s_i)) = \sum_{i=1}^{l} (1+ \sum_{j=0}^{k} a_j s_i^j) = l + \sum_{j=0}^{k} a_j \sum_{i=1}^{l} s_i^j \leq$$

$$\leq l + \sum_{j=0}^{k} a_j (\sum_{i=1}^{l} s_i)^j + a_0(l-1) \leq \phi_1(n) + n(a_0+1),$$

which completes the proof.

The corresponding "unmarked" results are obtained in each case by considering # to be a nonterminal instead of a terminal symbol. In addition, P is extended with productions of the form $A_a \rightarrow a\#$ and $A_a \rightarrow a$ with $a \in \Sigma_1$. I.e. let $\Delta = \{A_a \rightarrow a\# \mid a \in \Sigma_1\}$, $\Omega = \{A_a \rightarrow a \mid a \in \Sigma_1\}$, where the nonterminals A_a do not occur in $V_1 \cup V_2$. Finally, the control languages are concatenated (to the right) with $\overline{\Delta}\,\Omega$ and $\overline{\Delta}^*\Omega^*$, respectively. Even in the proof of closure under Kleene + this construction adds only a linear contribution to the time function. For the remaining families *LIN* and *POLY* the results follow in a simple way from the case Φ equals $POLY(k)$ □

Proposition 3.2. *Let Φ be a family of bounding functions equal to LIN, $POLY(k)$ or POLY. Then the families Φ_{RO} are closed under intersection with regular languages.*

Proof. The closure under intersection with regular languages has been shown in Chapter II for RCB/RO languages by means of the well-known "triple" construction. Here we use the same construction, however, with some minor modifications due to the fact that we have to deal with λRCB/RO grammars. Starting from a λRCB/RO grammar (G_1, C_1) and a deterministic finite automaton $(Q, \Sigma_R, \delta, q_0, F)$ which accepts the reversal of a regular language R this construction results in a λRCB/RO grammar (G, C) that generates $L(G_1, C_1) \cap R$. Here $G = (V, \Sigma, P, S)$ with $\Sigma = \Sigma_1 \cap \Sigma_R$ and $V = N \cup \Sigma$. N is the set of nonterminals defined as follows. N contains two new symbols S and Z $(S, Z \notin V_1)$ and all triples of the form (u, A, t) where $u, t \in Q$ and $A \in V_1$. To complete N we add a symbol A_a for every $a \in \Sigma_1$. The set P of productions of G is defined by

$$P = P_0 \cup P_F \cup P_E \cup P_\Sigma \cup \cup \{P_\pi \mid \pi \in P_1\}.$$

The control language of (G,C) is given by

$$C = P_0\sigma(C_1)\overline{P}_F P_E P_\Sigma^*,$$

where

$$P_0 = \{S \to Z(u,S_1,q_0) \,|\, u \in Q\},$$
$$P_F = \{A_a \to Z(u,a,t) \,|\, u = \delta(t,a),\ u \in F,\ a \in \Sigma_1\},$$
$$P_E = \{A_a \to a \,|\, a \in \Sigma\},$$
$$P_\Sigma = \cup\{P_a \,|\, a \in \Sigma\},$$

with, for every $a \in \Sigma_1$,

$$P_a = \{(p,a,q) \to a \,|\, p,q \in Q,\ \delta(q,a) = p\}.$$

The finite substitution $\sigma : P_1 \cup \overline{P}_1 \to 2^{(P \cup \overline{P})^*}$ is defined by $\sigma(\pi) = P_\pi$ and $\sigma(\overline{\pi}) = \overline{P}_\pi$ for each $\pi \in P_1$. The set P_π is defined for every $\pi = A \to \alpha$ in P_1 by

$$P_\pi = \{(p,A,q) \to t \,|\, p,q \in Q,\ t \in \tilde{\alpha}_p^q\}$$

where for every p,q in Q

$$\tilde{x}_p^q = \{(p,x_1,p_1)\ldots(p_{m-1},x_m,q) \,|\, p_i \in Q,\ 1 \le i \le m\},$$

Let (G_1,C_1) be a λRCB/RO grammar that is bounded by ϕ_1, where $\phi_1 \in POLY(k)$. Then (G,C) is a $POLY(k)$-bounded λRCB/RO grammar, since $T_{(G,C)}(n) \le 1+\phi_1(n)+1+1+(n-1) = \phi_1(n)+n+2$; cf. the definition of C. From this the corresponding statements for the families *LIN* and *POLY* follow immediately. □

Proposition 3.3. *Let Φ be a family of bounding functions equal to LIN, $POLY(k)$ or POLY. Then the following closure properties hold.*

(a) *The family $\Phi_{RO/B/f}$ is closed under substitution.*

(b) *The families Φ_{RO} are closed under λ-free context-free substitution.*

Proof. (a) Let $L_1 = L(G_1,C_1)$ be a λRCB/RO/B/f language and let σ be a λRCB/RO/B/f-substitution $\sigma : \Sigma_1 \to 2^{\Sigma^*}$. Next, let $\Sigma_1 = \{a_1,\ldots,a_n\}$ and for each $a \in \Sigma_1$, let (G_a,C_a) be a λRCB/RO/B/f grammar with $G_a = (V_a,\Sigma,P_a,S_a)$ such that $L(G_a,C_a) = \sigma(a)$. Assume that for every $a \in \Sigma_1$, $N_1 \cap V_a = \emptyset$ and that $N_{a_i} \cap N_{a_j} = \emptyset$ if $i \ne j$ for every $1 \le i,\ j \le n$. Define alphabets $\Delta = \{S_{a_1},\ldots,S_{a_n}\}$ and $\Omega = \{Z_{a_1},\ldots,Z_{a_n}\}$. Let T be the control set $\cup\{C_a \,|\, a \in \Sigma_1\}$, and $U = \{A \to \alpha \,|\, A \in N_1, \alpha \in (N_1 \cup \Omega)^+\}$. We use the isomorphism $i : V_1 \to N_1 \cup \Omega$ defined by

$$i(A) = A \qquad \text{for each } A \text{ in } N_1,$$
$$i(a) = Z_a \qquad \text{for each } a \text{ in } \Sigma_1$$

to define a homomorphism $h : P_1 \cup \overline{P}_1 \to U \cup \overline{U}$ as follows

$$h(A \to \alpha) = A \to i(\alpha),$$
$$h(\alpha \to A) = i(\alpha) \to A.$$

Now we can define the λRCB/RO/B/f grammar (G,C) which generates the language $\sigma(L_1)$ by $G = (V, \Sigma, P, S)$, where

- $V = \cup\{V_a \mid a \in \Sigma_1\} \cup N_1 \cup \Delta \cup \Omega \cup \{Z\} \cup \{A_a \mid a \in \Sigma\}$
- $P = \cup\{P_a \mid a \in \Sigma_1\} \cup h(P_1) \cup P_Z \cup \Theta \cup \Psi$ with
  $P_Z = \{Z_a \to Z\, S_a \mid a \in \Sigma_1\}$,
  $\Theta = \{A_a \to Za \mid a \in \Sigma\}$,
  $\Psi = \{A_a \to a \mid a \in \Sigma\}$
- $S = S_1$

and $C = h(C_1)\, P_Z^* T^* \overline{\Theta}^* \Psi^*$.

The proof is completed as follows. Let (G_1, C_1) be bounded by ϕ_1 where $\phi_1 \in POLY(k)$ and

$$\phi_1(p) = \sum_{j=0}^{k} a_j\, p^j$$

and let for all $a_i \in \Sigma_1$ the languages $\sigma(a_i)$ be bounded by ψ_i with $\psi_i \in POLY(k)$ and

$$\psi_i(p) = \sum_{j=0}^{k} b_{ij}\, p^j$$

where $1 \le i \le n$. Let F be a bounding function, $F \in POLY(k)$, determined by

$$F(p) = \sum_{j=0}^{k} b_j\, p^j$$

where $b_j = \max\{b_{ij} \mid 1 \le i \le n\}$. Let $v = a_{u(1)} \ldots a_{u(l)}$ with $l \ge 1$ and u a function from $\mathbb{N}^+$ to $\{1, \ldots, n\}$. Furthermore, let $w = w_1 \ldots w_l = \sigma(v)$ such that $w_s \in \sigma(a_{u(s)})$, $1 \le s \le l$. Now with $C = h(C_1)\, P_Z^* T^* \overline{\Theta}^* \Psi^*$ we can write

$$T_{(G,C)}(|w|) \le \phi_1(l) + l + \sum_{s=1}^{l} \psi_{u(s)}(|w_s|) + l + l$$
$$\le \phi_1(l) + \sum_{s=1}^{l} F(|w_s|) + 3l$$
$$\le \phi_1(l) + F(|w|) + b_0(l-1) + 3l.$$

The latter inequality is obtained by using the same method as in the proof of closure under marked Kleene +. With

$$l \leq \sum_{s=1}^{l} |w_s|$$

the result follows immediately.

(b) The construction for the proof of Proposition 3.3(b) differs only from the proof of 3.3(a) in the following details. The language L_1 is a λRCB/RO language and the substitution is a λ-free context-free substitution. The grammars (G_a, C_a) for $\sigma(a)$ are λRCB/RO grammars with $C_a = P_a^*$. As a matter of fact, we do not need a nonterminal Z which is therefore omitted. Consequently, Θ, Ψ, P_Z and Ω are left out of (G,C) and P is equal to $\cup\{P_a \mid a \in \Sigma_1\} \cup h(P_1)$. We define U as $\{A \to \alpha \mid A \in N_1, \alpha \in (N_1 \cup \Delta)^*\}$ and the isomorphism i is defined by $i : V_1 \to N_1 \cup \Delta$ with $i(A) = A$, for each $A \in N_1$ and $i(a) = S_a$, for each $a \in \Sigma_1$. As the control language C we take $h(C_1)T^*$. Now the final steps of the proof are analogous to the case of substitution.

If Φ equals *LIN* or *POLY*, then the result follows from $\Phi = POLY(k)$ as a corollary. □

In Chapter II, Definition 4.1 we introduced the weak Chomsky Normal Form (CNF). This definition can be adapted to time-bounded λRCB grammars in the obvious way. The time-bounded variant of Theorem II.4.4 reads as follows.

Proposition 3.4. *Let Φ be a family of bounding functions. If Φ is equal to LIN, POLY(k) or POLY, then for every $\Phi_{RN/B/f}$ grammar (G_0, C_0) there exists an equivalent $\Phi_{RN/B/f}$ grammar (G,C) in weak CNF.*

Proof. Let (G_0, C_0) be bounded by some $\phi_0 \in \Phi$. The first step consists of transforming this grammar into an equivalent grammar (G_1, C_1) without chain rules. This is effected by incorporating chain rules into the other non-chain rules, whereas $C_1 = T(C_0)$ for some nondeterministic generalized sequential machine mapping T; cf. Lemma II.4.3 for the details of this construction. Since $|T(x)| \leq |x|$ for each control word x, (G_1, C_1) will also be bound by ϕ_0. From this grammar we obtain the final grammar (G,C) by "splitting" each rule of (G_1, C_1) into smaller rules having a right-hand side of length less than or equal to two. This is achieved by the following construction. We assume that G_1 has no chain rules. Let $P_1 = \{\pi_1, \ldots, \pi_n\}$ be the set of productions of G_1 with $\pi_i = A_i \to B_{i,1} \ldots B_{i,m_i}$. Let P be constructed as follows. Starting with the empty set, adjoin every production of P_1 to P which has a right-hand side with a length smaller than three. Next, for every $\pi_i \in P_1$ with $m_i \geq 3$ construct $m_i - 1$ new productions from this production as follows. Take $\pi_{i,1} = A_i \to B_{i,1} D_{i,1}$, $\pi_{i,2} = D_{i,1} \to B_{i,2} D_{i,2}$, ..., $\pi_{i,m_i-1} = D_{i,m_i-2} \to B_{i,m_i-1} B_{i,m_i}$. We assume that the $D_{i,j}$'s are distinct from

each other, and that these $D_{i,j}$'s constitute the set D. The productions $\pi_{i,j}$ will be adjoined to P. Now we define a homomorphism $h : P_1 \to P^*$ with $h(\pi_i) = \pi_i$ if $m_i \leq 2$ and $h(\pi_i) = \pi_{i,1} \ldots \pi_{i,m_i-1}$ if $m_i \geq 3$. Furthermore, for a reduction $\bar{\pi} \in \bar{P}_1$ define $h(\bar{\pi}) = \overline{h(\pi)}$, using $\overline{\pi\tau} = \bar{\tau}\bar{\pi}$ for every $\pi, \tau \in P_1$. Finally, we take $C = h(C_1)$ and $G = (V_1 \cup D, \Sigma_1, P, S_1)$. Now let M be the maximum value of the length of a right-hand side of a rule of (G_1, C_1). Then we have $T_{(G,C)}(n) \leq (M-1)\phi_0(n)$ if $M \geq 3$ and $T_{(G,C)}(n) \leq \phi_0(n)$ otherwise. Hence (G, C) is bounded by $(M-1)\phi_0$. This completes the proof. □

4. Parsing λRCB Languages.

In this section we present depth-first bottom-up parsing algorithms for some Φ_m languages where Φ is a family of bounding functions. Although the algorithms are modifications of a well-known backtrack algorithm, the presence of reductions introduces some principal differences when compared with the usual bottom-up parsing algorithms for context-free languages. In the "normal case" of bottom-up parsing, a correct sequence of productions which rewrites S into a string w is determined by applying reduce and shift operations to the input string w. In our framework, where reductions may occur in the control language, we also ought to apply *produce operations*. This means that a reduction $\alpha \to A$ in the control language causes the parsing algorithm to rewrite the right-most nonterminal of the current sentential form into α, at least if this right-most nonterminal is equal to A. We say that a rule j is applicable (with respect to the parsing algorithm) to a string α if there is a string β such that $app_m(\bar{j}, \alpha, \beta)$ (assuming, of course, that $\bar{\bar{\pi}} = \pi$, for each π in P). In other words, a production in the control language will cause a reduce operation at the parsing process; a reduction in the control language will cause a produce operation. The presence of reductions has also the effect that we cannot use lookahead to obtain faster algorithms, at least not in a straightforward way as in the case of ordinary context-free parsing. This can be illustrated by the following observation, concerning the RN-mode. A produce operation rewrites a nonterminal A into a string α according to a reduction $\alpha \to A$ in the control language. In this case, the longest postfix of α which consists entirely of terminals ought to be considered as a string of terminals that have not yet been involved in the parsing algorithm by shift operations.

All algorithms in this section are bottom-up parsers. This is due to the fact that in RCB grammars we rewrite the right-most nonterminal, i.e., we consider right-most derivations. In case of the corresponding "LN-mode" (Left Nonterminal) a top-down parser would be needed. First we present a

parsing algorithm for the mode RN/B/f. The algorithm is inspired by the depth-first bottom-up parsing algorithm presented in [Sud]. As in [Sud], we use a stack (here represented by **T**) to handle the backtrack information.

Algorithm 4.1. *A depth-first bottom-up parser for λRCB/RN/B/f languages.*

input: – λ-free RCB/RN/B/f grammar (G,C) represented by a λ-free context-free grammar $G = (V, \Sigma, P, S)$, and a deterministic finite automaton $M = (Q, \Delta, \delta, q_0, F)$, with $\Delta \subseteq P \cup \overline{P}$, that accepts C^R, i.e., the reverse of C.
– string $w \in \Sigma^*$, where $w = w_1 \ldots w_n$, $n \geq 1$, $w_i \in \Sigma$.
– bounding function ϕ.

output: – a control word (a parse) c with c deriving w from S if such a c in C exists, otherwise a reject message.

1. $K := \phi(n)$
   PUSH($[\lambda, w, 0, 0, \lambda, q_0]$, **T**)

2. **repeat**
   $[u,v,i,t,c,q] :=$ POP(**T**)
   dead_end := false
   repeat
   Find the first rule j with $j > i$ that satisfies
   i) $j \in Follow(q)$
   ii) $j = xAy \rightarrow z$ with $u = pz$ and $x,p \in V^*$, $z \in V^+$, $y \in \Sigma^*$
   if there is such a j **then**
   PUSH($[u,v,j,t,c,q]$, **T**)
   $u := pxAy$
   $rearrange(u,v)$
   $i := 0$
   $t := t+1$
   $q := \delta(q,j)$
   $c := jc$
   end if
   if there is no such j **then**
   if $v \neq \lambda$ **then**
   shift(u,v)
   $i := 0$
   else
   dead_end := true
   end if
   end if

until ($u = S$ and $v = \lambda$ and $q \in F$) or dead_end or $t = K$
until ($u = S$ and $v = \lambda$ and $q \in F$) or EMPTY(**T**)

3. **if** EMPTY(**T**) **then** reject **else** output(c) □

The algorithm works as follows. As already stated, a stack **T** is used to manage the information where to continue with the parsing algorithm in case we have to backtrack from a wrong parsing decision. To this end each element of the stack consists of six items. The first and second item are strings from V^* which constitute − when concatenated − the string on which the latest rule has been applied. The first item is associated with the variable u and the second with the variable v. The algorithm is organized in such a way that, after each operation on u, the pair (u,v) is rearranged (if necessary) into the pair (u',v') such that $u'v' = uv$, $u' \in V^*(V-\Sigma)$ and $v \in \Sigma^*$. Throughout this section, we suppose that this rearranging is performed by a procedure *rearrange* (u,v). So the variable v contains a string from Σ^* during the entire parsing process. This string v represents more or less the input which has not yet been processed. Because we also have to deal with reductions in the control language, v may even become longer during the parsing process. This happens in case a nonterminal A at the right side of u is rewritten to a string with terminals at the right side, according to the application of some reduction in the control word from C. After the PUSH operation, these terminals are adjoined to the left side of v. As already mentioned, this is performed by *rearrange* (u,v). The sixth item, associated with the variable q, is a state of the deterministic finite automaton M. With each state s we associate a set *Follow* (s) which is defined by

$$Follow(s) = \{i \in P \cup \bar{P} \mid \exists p.\, \delta(s,i) = p\},$$

i.e., this set is formed by all label names of the outgoing arcs of the state s. The third item, associated with the variable i, gives us the index of the latest rule which has been tried. We represent each rule from $P \cup \bar{P}$ by a number from $1 \ldots 2 \cdot |P|$. Then i indicates that the next rule that will be tried, ought to have an index greater than i. If $i = 0$, then no rules have yet been tried after entering the state q. The fourth item of a stack element is associated with t. It stands for the number of rules used so far at the current path, and it is increased by one each time a rule can be applied. If t becomes equal to the time-bound K, no rules will be tried any more. If the stack is not empty at that moment, then we backtrack by popping an element from the stack, which will have an item t with $t < K$. Finally, the fifth item, associated with the variable c, contains the parse string, and after a successful parse of an input string c equals a control word from C which derives w.

The algorithm starts with calculating the time-bound K from ϕ and n, the length of the input w. The stack is initiated by pushing $[\lambda, w, 0, 0, \lambda, q_0]$ onto the stack. The body of the algorithm begins with popping an element $[u,v,i,t,c,q]$ from the stack **T**. Starting at $j = i+1$ we try to find the first j smaller or equal to $2 \cdot |P|$ with $j \in Follow(q)$ and j is the index of a rule applicable to u with respect to the parsing algorithm. If this search is successful, then we first put backtrack information onto the stack by PUSH($[u,v,j,t,c,q]$,**T**). Then we perform a reduce or produce operation on the string u, according to the type of the rule associated with j, obtaining a new string u'. We change q to the new state q' of M which is equal to $\delta(q,j)$, and set i equal to zero. Next we increase the counter t by one, and the index j is adjoined to the left of the old string c. We obtain a new "input string" v' differing from the old string v in case we applied a produce operation $B \to xAy$ with $y \in \Sigma^+$. This is effected by $rearrange(u,v)$. If there exists no rule with index $j > i$ and $j \in Follow(q)$ with j applicable to u, then we shift one terminal symbol a from the remaining input v to the right of u in case $v \neq \lambda$. Hereafter we try repeatedly to find a proper rule which is applicable to the new string ua. If $v = \lambda$, then we have to backtrack, which is effected by chancing the value of the variable *dead_end* to true.

Let M be a deterministic finite automaton with a set of states Q. Then we define **M** by

$$\mathbf{M} = \max\{Card(Follow(q)) \mid q \in Q\}.$$

where for a set B, $Card(B)$ denotes its cardinality.

Proposition 4.2. *Let (G,C) be a λRCB/RN/B/f grammar bounded by a bounding function ϕ and let w be a string from Σ^+ with $n = |w|$. Then Algorithm 4.1 can decide in time $O(\mathbf{M}^{\phi^2(n)})$ and in space $O(\phi^2(n))$ whether or not w is an element of $L(G,C)$. If $w \in L(G,C)$, then the algorithm produces also a control word c deriving w.*

Proof. Suppose $w \in \Sigma^+$. Because the algorithm cuts off every possible derivation with a length greater than $\phi(|w|)$ it has to search among a finite number of strings from $(P \cup \bar{P})^*$. Furthermore, by Lemma 2.7 the existence of a control word $c \in C$ with length smaller than or equal to $\phi(|w|)$ is guaranteed in case $w \in L(G,C)$. So the algorithm can decide in a bounded amount of time and space whether or not $w \in L(G,C)$. To be more precise, if we count every PUSH operation as one unit of time we obtain the time and space bounds stated above as follows. The stack will have a height of at most $\phi(n)$ elements. Each element will need an amount of space proportional to $\phi(n)$ because once we have recognized a nonterminal A, it is possible that this nonterminal will be rewritten by a series of reductions $\alpha A \to A$ in the control language, at most $\phi(n)-1$ times, where $|\alpha| <$

$\max\{|\gamma| \mid A \to \gamma \in P\}$. Summarizing, the algorithm will need at most $O(\phi^2(n))$ units of space. At every node q of M, where M is the deterministic finite automaton of Algorithm 4.1, the algorithm can make at most **M** wrong tries after each shift operation. The expected number of shift operations is proportional to $\phi(n)$. This is due to the same reason that a stack element has an $O(\phi(n))$ need of space. Then at each node we can perform at most $O(\mathbf{M}^{\phi(n)})$ PUSH actions which finally lead to a dead alley situation. So there exist at most $O(\mathbf{M}^{\phi^2(n)})$ control words the algorithm ought to check before terminating. □

Algorithm 4.1 presented above serves as a base for other parsers. Depending on the mode *m*, we modify Algorithm 4.1 in order to obtain parsers for λRCB/*m* languages. We will discuss parsers for the modes RN/B/g, RN/S/f and RO/B/f in some detail. Further modifications – yielding parsers for the remaining modes – are left to the reader as an exercise.

The algorithm for λRCB/RN/B/g languages can be obtained from Algorithm 4.1 by chancing the part beginning at "ii) $j = xAy \to z$ with $u = pz$ and . . ." up to and including "$u := pxAy$" into the following sequence of instructions.

ii) $j = xAy \to z$ with $u = pz$ and $x, p \in V^*$, $z \in V^+$, $y \in \Sigma^*$
or $j = x \to A$ with $u = pA$ and $x \in \Sigma^+$, $p \in V^*$
if there is such a j **then**
PUSH($[u, v, j, t, c, q]$, **T**)
if j is a general reduction **then**
(* $j = x \to A$, $x \in \Sigma^+$ *)
$u := px$
else
$u := pxAy$
end if

Concerning the time and space complexity, we can easily show that for the algorithm for λRCB/RN/B/g languages these will be of the same order as for Algorithm 4.1. This fact indicates that the upper bounds presented in Proposition 4.2 are probably not very tight. Cf. also the remark on the complexity of the λRCB/RO/B/f-parser at the end of this section.

Next we consider a parsing algorithm for the RN/S/f-mode; cf. Algorithm 4.3 below. If we compare this algorithm with Algorithm 4.1, then the following differences are conspicuous. Stack elements have been extended with a seventh and an eighth item. The seventh item will contain the value of a boolean variable *Skip*. *Skip* indicates whether the algorithm ought to skip a rule of the control language. If *Skip* = false then we execute the same lines as in the algorithm for the RN/B/f-mode (plus the initializing of the

eighth item, *notapp*). However, at some moment, if no rule j with $j > i$ is applicable after shifting the entire remaining input string, we can try to skip a rule. Therefore we replace "dead_end := true" from Algorithm 4.1 by "*Skip* := true; i := 0". To keep the administration concerning which rule is not applicable in the context of the sentential form uv and the state q of the deterministic finite automaton M, we use the variable *notapp*. It denotes a subset of $P \cup \overline{P}_f$, where

$$\overline{P}_f = \{\alpha \to A \mid A \to \alpha \in P,\ \alpha \in V^* - \Sigma^*\}.$$

Each time a new state q' is computed from $\delta(j,q)$, *notapp* is set to the value $P \cup \overline{P}_f$, which is also the initial value of *notapp*. After finding an applicable rule j we remove this rule from *notapp*. This is effected by storing this fact, together with the other backtrack information, in the eighth item of the stack element by PUSH($[u,v,j,t,c,q,Skip,notapp - \{j\}]$, **T**).

Algorithm 4.3. *A depth-first bottom-up parser for λRCB/RN/S/f grammars.*

input: – λ-free RCB/RN/S/f grammar (G,C) represented by a λ-free context-free grammar $G = (V, \Sigma, P, S)$, and a deterministic finite automaton $M = (Q, \Delta, \delta, q_0, F)$, with $\Delta \subseteq P \cup \overline{P}$, that accepts C^R, i.e., the reverse of C.
– string $w \in \Sigma^*$, where $w = w_1 \ldots w_n$, $n \geq 1$, $w_i \in \Sigma$.
– bounding function ϕ.

output: – a control word (a parse) c with c deriving w from S if such a c in C exists, otherwise a reject message.

1. $K := \phi(n)$
   PUSH($[\lambda, w, 0, 0, \lambda, q_0, \text{false}, P \cup \overline{P}_f]$, **T**)

2. **repeat**
   $[u,v,i,t,c,q,Skip,notapp]$:= POP(**T**)
   dead_end := false
   repeat
   if not *Skip* **then**
   Find the first rule j with $j > i$ that satisfies
   i) $j \in Follow(q)$
   ii) $j = xAy \to z$ with $u = pz$
   and $x,p \in V^*$, $z \in V^+$, $y \in \Sigma^*$
   if there is such a j **then**
   PUSH($[u,v,j,t,c,q,Skip,notapp - \{j\}]$, **T**)
   $u := pxAy$
   rearrange(u,v)
   $i := 0$

$t := t+1$
$q := \delta(q,j)$
$notapp := P \cup \bar{P}_f$
$c := jc$
end if
if there is no such j **then**
$i := 0$
if $v \neq \lambda$ **then**
shift(u,v)
else
$Skip :=$ true
end if
end if
else (* $Skip$ = true *)
Find the first rule j with $j > i$ that satisfies
i) $j \in Follow(q)$
ii) $j \in notapp$
if there is such a j **then**
$rearrange(u,v)$
PUSH($[u,v,j,t,c,q,Skip,notapp]$,**T**)
$i := 0$
$q := \delta(q,j)$
$notapp := P \cup \bar{P}_f$
$Skip :=$ false
else
dead_end := true
end if
end if
until ($u = S$ and $v = \lambda$ and $q \in F$) or dead_end or $t = K$
until ($u = S$ and $v = \lambda$ and $q \in F$) or EMPTY(**T**)

3. **if** EMPTY(**T**) **then** reject **else** output(c) □

So after setting the variable *Skip* to true in the **then**-part of the "**if** not *Skip* **then** ... **else** ..." statement, we will enter the next turn of the inner repeat loop the **else**-part of the "**if** not *Skip* **then** ... **else** ..." statement. Because we have set i equal to 0 we can try each rule j that is not applicable at the current string uv. If we find such a j, then we first ought to perform $rearrange(u,v)$. This is because v may be equal to λ due to shift operations. Then we store these new u and v together with the other back-track information $(j,t,c,q,Skip,notapp)$ by pushing them onto the stack **T** (where *Skip* has the value true). The variable i is set to 0, *Skip* to false and

we compute the new state q' by $\delta(q,j)$. Furthermore, in this new context consisting of uv and q', *notapp* is initialized by $P \cup \overline{P}_f$. Of course, no rule can be concatenated to the control string already found. Also the time counter t will not be increased. If there are no rules left that are not applicable, this path has been exhausted and we have reached a dead end situation.

Algorithm 4.3 can make at each node q of M at most $2 \cdot \mathbf{M}$ wrong decisions after each shift operation. This results in a time complexity of $O((2 \cdot \mathbf{M})^{\phi^2(n)})$. The space complexity is of the same order as in Algorithm 4.1; cf. the proof of Proposition 4.2.

As a last example of λRCB/m-parsers we discuss the case in which m is equal to RO/B/f. In this mode, rules can be applied more freely than in the mode RN/B/f. This means that we ought to weaken the corresponding condition in Algorithm 4.1. Viz., we change

ii) $j = xAy \rightarrow z$ with $u = pz$ and $x,p \in V^*, z \in V^+, y \in \Sigma^*$

into

ii) $j = xAy \rightarrow z$ with $u = pzs$ and $x,p,s \in V^*, z \in V^+, y \in \Sigma^*$

and either (($x = \lambda$ and $y = \lambda$) and A does not occur in s)

or (($x \neq \lambda$ or $y \neq \lambda$) and z does not occur in s)

In addition, we change "$u := pxAy$" from Algorithm 4.1 into "$u := pxAys$".

The time and space complexity of this modified algorithm is of the same order as in Algorithm 4.1. This is due to not taking into account the time needed to check for the applicability of a rule j from $Follow(q)$. This latter test is expressed in condition ii) occurring in the various algorithms. It is just this condition that depends on the mode under consideration.

In the algorithms presented above, some improvements are possible. Viz. we do not need to push backtrack information onto the stack if it happens that $Follow(q)$ possesses only one element. Furthermore, for each pair u and v just popped from the stack, we observe that, once we have shifted from v to u, we do not need to check for the applicability of reductions from $Follow(q)$. Another improvement is the following. It is possible for a state q that all productions in $Follow(q)$ are fair productions, i.e., their right-hand side is an element of $V^*(V-\Sigma)V^*$. Then after a (bounded) number of shift operations, depending on the set $Follow(q)$, no further shift operations are needed. This is because the length of the longest postfix, consisting of terminals only, of the right-hand side of a production π has a maximal value on the set $Follow(q)$. In the same way, whenever there are also terminal

productions in $Follow(q)$, we need only to check for the applicability of terminal productions on the intermediate string uv (with respect to the parsing algorithm) after a bounded number of shift operations.

These possible improvements show that the derived upper bounds for the time and space complexity are probably not very tight. Thus it is likely that a more careful analysis will yield better upper bounds for the improved parsing algorithms.

5. Concluding Remarks.

In this chapter we extended the idea of time-bounded grammars, as introduced in [Boo71, Gla], to the concept of λRCB grammar. We showed that for the mode RN/B/f we have $\Phi_{RN/B/f} = \lambda RCB/RN/B/f = \lambda$CFL, where Φ is equal to *POLY*, $POLY(k)$, or *LIN*. We also constructed parsers for some of the modes. In Table 1 we summarize the closure properties established in Section 3. In this table, an entry which is empty indicates an open problem; a plus means a positive result.

Closure properties of Φ_m languages with Φ equal to *POLY*, $POLY(k)$ or *LIN*.

	RN				RO			
	B		S		B		S	
	f	g	f	g	f	g	f	g
union	+	+	+	+	+	+	+	+
concatenation	+				+		+	
marked concatenation	+		+		+		+	
Kleene +	+				+			
marked Kleene +	+		+		+			
intersection with a regular set	+				+	+	+	+
λ-free context-free substitution	+				+	+	+	+
substitution	+				+			

Table 1.

Note that the positive results for the mode RN/B/f are due to the fact that λCFL = $\Phi_{RN/B/f}$, with Φ as above.

The closure properties for Φ_m languages, with Φ equal to *POLY*, $POLY(k)$ or *LIN*, can also be established for other language families based on more general control languages and on less restricted families of

μ	$\forall p \geq 2 . \forall \psi_1, \ldots, \psi_p \in \Phi . \exists \phi \in \Phi . \forall n . \phi(n) \geq \max\{\psi_i(n) \mid 1 \leq i \leq p\}$
μ_2	$\forall \psi_1, \psi_2 \in \Phi . \exists \phi \in \Phi . \forall n . (\phi(n) \geq \max\{\psi_1(n), \psi_2(n)\}$
α	$\forall \phi, \psi \in \Phi . \lambda n . (\phi(n) + \psi(n)) \in \Phi$
α_0	$\forall \phi \in \Phi, \forall c \geq 0 . \lambda n . (\phi(n) + c) \in \Phi$
α_1	$\forall \phi \in \Phi, \forall c, d \geq 0 . \lambda n . (\phi(n) + d \cdot n + c) \in \Phi$

c and d are natural numbers.

Table 2.

bounding functions. Let **C** denote an arbitrary family of control languages and Φ an arbitrary family of bounding functions. Then for each closure property it is possible to list simple properties of **C** and of Φ which imply a certain closure property of the family of languages generated by Φ-bounded **C**-controlled grammars. Results of this type – which can easily be proven in a way similar to the proofs in Section 3 – are in Table 3.

Assumptions on **C**	Assumptions on Φ	Closure property of $\Phi_m(\mathbf{C})$
marked union	μ_2, α_0	union
concatenation, left and right-marking	α_0, α	concatenation
concatenation, left-marking	α_0, α	marked concatenation
concatenation, Kleene +, Kleene $*$, left-marking	α_1	Kleene +
concatenation, Kleene $*$, left-marking	α_1	marked Kleene +
union, concatenation, Kleene $*$, reversal, finite substitution	α_1	intersection by a regular set
union, concatenation, Kleene $*$, homomorphism	μ, α	λ-free context-free substitution
union, concatenation, Kleene $*$, homomorphism	μ, α, α_1	substitution

Table 3.

The meaning of the assumptions on the family of bounding functions Φ mentioned in Table 3 are listed in Table 2. For a precise definition of

closure under left and right-marking we refer to Section II.6. With each closure property mentioned in the table a specific set of modes is necessary to obtain a proper result. This set can be found in the corresponding proposition from Section 3. Since **C** now replaces the family of regular languages, we are dealing with λ-free **C**-controlled bidirectional grammars (λ**C**CB grammars) rather than λRCB grammars. Then $\Phi_m(\mathbf{C})$ denotes the family of languages generated by λ**C**CB/*m* grammars that are bounded by some bounding function from Φ. For an arbitrary family **C** of control languages, Lemma 2.8 and Corollary 2.10 no longer hold. Under which conditions on **C**, the family $\Phi_{RN/B/f}(\mathbf{C})$ − with Φ equal to *POLY*, $POLY(K)$ or *LIN* − shares all closure properties of the family λ*CFL*, remains therefore an open problem. Note that Table 3 only provides a partial answer to this question.

CHAPTER IV

Generating Power of RCB/RO Grammars

1. Introduction

One aspect of the derivational process in RCB grammars is the selection of the terminal that has to be rewritten – if possible – by the next rule prescribed by the control word. In Chapter II the right-occurrence or RO-mode has been introduced. We introduced this rather "exotic" way of rewriting in order to establish some closure properties of the corresponding family of RCB languages, viz. closure under homomorphism, inverse homomorphism, intersection with a regular set, and under context-free substitution. Now the main result of this chapter is, that if the mode of derivation *m* includes this RO-mode instance, then the resulting language family equals the family of recursively enumerable languages. And so this family inherits all (closure) properties of the family of recursively enumerable languages. But the proofs in Section II.3 for the RO-mode remain to have some interest since they deal with rules rather than with productions only.

This chapter is organized in the following way. In Section 2 we recall some definitions concerning Turing machines and related concepts in order to fix our notation. Section 3 is devoted to the proof the main result concerning the generating power of RCB grammars provided with the RO-mode. Some consequences of this result are mentioned in Section 4; viz. the time-bounded RCB/RO grammars of Chapter III are weaker than ordinary RCB/RO grammars with respect to generating power. This follows from the fact that time-bounded RCB languages are recursive; cf. Chapter III. Then in Section 5 we discuss the difference between the RS-mode and RA-mode introduced in Chapter I, and the RN-mode and RO-mode introduced in Chapter II. We show that the RA-mode has the same generating power as the RO-mode. Finally, Section 6 contains some concluding remarks and open problems.

2. Preliminaries

We refer to [Har, HopUll79] for all unexplained notations and concepts from formal languages and complexity theory. Another useful standard text is [Sal73]. First, we recall some basic definitions and terminology with respect to Turing machines.

Definition 2.1. A *deterministic single-tape Turing machine* is a 7-tuple $A = (Q, \Sigma, \Gamma, B, \delta, q_0, F)$, where

- Q is a finite nonempty set of *states*,
- Σ is a finite nonempty set of *input symbols*,
- Γ is a finite nonempty set of *work symbols* and $\Sigma \subseteq \Gamma$,
- $B \in \Gamma - \Sigma$ is the *blank symbol*,
- $q_0 \in Q$ is the *initial state*,
- $F \subseteq Q$ is the set of *final* or *accepting* states,
- δ is a partial mapping from $Q \times \Gamma$ into $Q \times \Gamma \times \{-1, 0, 1\}$. This mapping is called the *transition function*. □

From the so-called instantaneous description of a Turing machine A we can infer in what state A is, the contents of its tape, and the head position on the tape. We assume $Q \cap \Gamma = \emptyset$.

Definition 2.2. An *instantaneous description* or ID of a deterministic single-tape Turing machine A equal to $(Q, \Sigma, \Gamma, B, \delta, q_0, F)$ is any element of $\Gamma^* Q \Gamma^+$. An *initial* ID is an ID of the form $q_0 w$ with $w \in \Sigma^+ \cup \{B\}$ and an *accepting* ID is any element of $\Gamma^* F \Gamma^+$. □

In an ID $\alpha q \beta$, the symbol q represents the state in which the Turing machine is. The string $\alpha\beta$ denotes the contents of the tape such that the head is scanning the first symbol of β.

Definition 2.3. Let $A = (Q, \Sigma, \Gamma, B, \delta, q_0, F)$ be a deterministic single-tape Turing machine. The *transition relation* $\vdash$ on $\Gamma^* Q \Gamma^+$ is defined as follows. Let x, y be ID's, where $x = \alpha aqb\beta$ and $y = \alpha' q' \beta'$ with $\alpha a, \alpha' \in \Gamma^*$, $a \in \Gamma \cup \{\lambda\}$, and $b\beta, \beta' \in \Gamma^+$. Furthermore, let $\delta(q, b) = (p, c, d)$. Then A rewrites b into c and moves one position to the right [left] if $d = +1$ [-1, respectively], and if $d = 0$ the position of the head does not change. Now we write $x \vdash y$ if and only if

- $p = q'$ and
- ($d = +1$ and $\alpha' = \alpha ac$ and $\beta' = \beta$) or
  ($d = -1$ and $\alpha' = \alpha$ and $\beta' = ac\beta$) or
  ($d = 0$ and $\alpha' = \alpha a$ and $\beta' = c\beta$).

As usual, $\vdash^*$ denotes the reflexive and transitive closure of $\vdash$. □

Definition 2.4. Let $A = (Q, \Sigma, \Gamma, B, \delta, q_0, F)$ be a deterministic single-tape Turing machine and $w \in \Sigma^+ \cup \{B\}$. The Turing machine A *accepts* w (when $w \in \Sigma^+$) or A *accepts* λ (when $w = B$) if

$$q_0 w \vdash^* \alpha q \beta \text{ for some } q \in F.$$

The set of all w in Σ^* accepted by A is called the *language* accepted by A; it is denoted by $T(A)$. Thus $T(A) = \{w \in \Sigma^* \mid A \text{ accepts } w\}$.

A language L_0 is called *recursively enumerable*, if $L_0 = T(A)$ for some deterministic single-tape Turing machine A. The family of recursively enumerable languages is denoted by RE. □

It is well known [Har, HopUll79, Sal73] that the family of recursively enumerable languages is equal to the family of Chomsky type-0 languages or phrase-structure languages.

3. The Main Result

The proof of Proposition 3.1 has been inspired by the proof of Lemma 9.5.2 in [Har] which establishes the equality of the family of phrase-structure languages and the family of the recursively enumerable languages. In that proof some arbitrary phrase-structure productions rather than context-free productions play of course an essential part. In order to show that for certain modes m, RCB/m grammars are able to generate all recursively enumerable languages we have to simulate arbitrary phrase-structure productions by a combination of context-free productions and reductions. The idea of the proof below is that we simply replace each of these phrase-structure productions by a reduction immediately followed by a production such that these two rules have the same effect as that single phrase-structure production.

For each mode m, let $\mathbf{L}_m$ denote the family of languages generated by RCB/m grammars.

Proposition 3.1. *A language L_0 is an RCB/RO language if and only if L_0 is recursively enumerable. Equivalently,* $\mathbf{L}_{RO} = RE$.

Proof. Let L_0 be equal to $T(A)$, the set of strings in Σ^* accepted by the deterministic single-tape Turing machine A, where $A = (Q, \Sigma, \Gamma, B, \delta, q_0, F)$. Furthermore, we assume that $\delta(q, a) = \varnothing$ for each q in F. First, we construct an RCB grammar (G, C) with $G = (V, \Sigma \cup \{\$\}, P, S)$ such that $L_{RO}(G, C) = \{\$\}L_0$. This RCB grammar (G, C) starts with producing nondeterministically a coded version of a word x in Σ^*. Then it simulates the computation of A on input x. In case this simulated computation of A on input x reaches a final state, then (G, C) will yield $\$x$ as the string it generates.

We define the alphabet V of G by

$$V = \Sigma \cup \{\$\} \cup V_0 \cup V_1 \cup V_2 \cup V_3 \cup \{S, U, W_\$, W\} \cup Q$$

where

$$V_0 = (\Sigma \cup \{\lambda\}) \times \Gamma,$$
$$V_1 = Q \times (\Sigma \cup \{\lambda\}) \times \Gamma,$$
$$V_2 = (\Sigma \cup \{\lambda\}) \times \Gamma \times Q \times (\Sigma \cup \{\lambda\}) \times \Gamma,$$
$$V_3 = \{W_a \mid a \in \Sigma\}.$$

The set P is the union of a finite number of mutually disjoint sets, each of which consists of a finite number of productions. This subdivision of the elements of P facilitates the description of the way in which (G, C) simulates the computations according to A.

The subsets $\{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5\}$, $P_{\Sigma\Sigma}$, $P_{\$R}$ and $P_{\$L}$ of P consist of productions that initialize the simulation of the Turing machine A. These productions are defined by

$$\pi_1 = S \to S(\lambda, B), \qquad \pi_2 = S \to W_\$ U(\lambda, B),$$
$$\pi_3 = U \to \$, \qquad \pi_4 = W \to W_\$ \$,$$
$$\pi_5 = W \to \$q_0.$$

Furthermore,

$$P_{\Sigma\Sigma} = \{U \to (a,a)U \mid a \in \Sigma\},$$
$$P_{\$R} = \{W_a \to (a,a)\$ \mid a \in \Sigma\},$$
$$P_{\$L} = \{W_a \to \$(a,a) \mid a \in \Sigma\}.$$

In the next six subsets of P – to be defined below – the set $P_{i,I}$ $(i = -1, 0, 1)$ consists of the productions that are necessary to start a simulation of an i-step of the Turing machine A. In fact, only reductions from $\overline{P}_{i,I}$ will be used. Then the rules in the corresponding set P_i will actually complete that simulation.

$$P_{0,I} = \{(p,a,D) \to p(a,D) \mid a \in \Sigma \cup \{\lambda\},\ p \in Q,\ D \in \Gamma,$$
$$\exists E \in \Gamma, \exists q \in Q \,.\, \delta(p,D) = (q,E,0)\},$$
$$P_0 = \{(p,a,D) \to q(a,E) \mid a \in \Sigma \cup \{\lambda\},\ p,q \in Q,\ D,E \in \Gamma,\ \delta(p,D) = (q,E,0)\},$$
$$P_{1,I} = \{(p,a,D) \to p(a,D) \mid a \in \Sigma \cup \{\lambda\},\ p \in Q,\ D \in \Gamma,$$
$$\exists E \in \Gamma, \exists q \in Q \,.\, \delta(p,D) = (q,E,1)\},$$
$$P_1 = \{(p,a,D) \to (a,E)q \mid a \in \Sigma \cup \{\lambda\},\ p,q \in Q,\ D,E \in \Gamma,\ \delta(p,D) = (q,E,1)\},$$

$$P_{-1,I} = \{(b,H,p,a,D) \to (b,H)\,p\,(a,D) \mid a,b \in \Sigma \cup \{\lambda\},\ p \in Q,\ D,H \in \Gamma,$$
$$\exists E \in \Gamma, \exists q \in Q \,.\, \delta(p,D) = (q,E,-1)\},$$
$$P_{-1} = \{(b,H,p,a,D) \to q\,(b,H)(a,E) \mid a,b \in \Sigma \cup \{\lambda\},\ p,q \in Q,$$
$$D,E,H \in \Gamma,\ \delta(p,D) = (q,E,-1)\}.$$

Once we reach a final state in the simulation of the Turing machine A, the next four subsets of P take care of generating the terminal string that has apparently been accepted by (the simulation of) the Turing machine.

$$P_R = \{(q,a,D) \to q\,(a,D) \mid q \in F,\ a \in \Sigma \cup \{\lambda\},\ D \in \Gamma\},$$
$$P_L = \{(q,a,D) \to (a,D)\,q \mid q \in F,\ a \in \Sigma \cup \{\lambda\},\ D \in \Gamma\},$$
$$P_\Sigma = \{(q,a,D) \to aq \mid q \in F,\ a \in \Sigma \cup \{\lambda\},\ D \in \Gamma\},$$
$$P_\lambda = \{q \to \lambda \mid q \in F\}.$$

Finally, we define the control language C of (G,C) by $C = (P \cup \bar{P})^*$.

A consequence of the equality $C = (P \cup \bar{P})^*$ is that the generating power of the B and S-mode will be equal. This is due to the fact that if we have some control string c in C such that $S \Rightarrow^{c}_{RO/S/f} w$, then the string c' obtained from c by removing each skipped rule has the property $S \Rightarrow^{c'}_{RO/B/f} w$ and $c' \in C$.

The construction sketched above works as follows. If the Turing machine A accepts the string $a_1 \ldots a_n$, then it will stop after a finite computation. During this computation A uses, apart from the n cells on which the input has been written, some number of additional cells – say k $(k \geq 0)$ – to the right of the input. Now we can only start a derivation of (G,C) by applying k times $(k \geq 0)$ the production $\pi_1 = S \to S(\lambda,B)$ to S, followed by π_2 in order to remove S. This production is followed by zero or more applications of productions of the form $U \to (a,a)U$ with $a \in \Sigma$, and a single application of the production $U \to \$$. Thus there exists a control string c_1 in $\{\pi_1\}^*\{\pi_2\}P^*_{\Sigma\Sigma}\{\pi_3\}$ such that

$$S \Rightarrow^{c_1}_{RO/f} W_\$(a_1,a_1)\ldots(a_n,a_n)\$(\lambda,B)^k, \qquad (n+k \geq 1).$$

The string obtained by this subderivation will be denoted by $\alpha_{n,k}$.

By zero or more applications of pairs of the form $(a,a)\$ \to W_a$ and $W_a \to \$(a,a)$ with $a \in \Sigma$, and followed by the application of $\bar{\pi}_4$ and π_5 we observe that there exists a control string c_2 in $(\bar{P}_{\$R}P_{\$L})^*\{\bar{\pi}_4\pi_5\}$ such that

$$\alpha_{n,k} \Rightarrow^{c_2}_{RO/f} \$q_0(a_1,a_1)\ldots(a_n,a_n)(\lambda,B)^k, \qquad (n+k \geq 1). \qquad (1)$$

The string obtained by this subderivation will be denoted by $\omega_{n,k}$.

Note that inserting productions and reductions from $P_{\Sigma\Sigma} \cup P_{\$R} \cup P_{\$L}\{\pi_1,\pi_2,\pi_3,\pi_4,\pi_5\}$ in c_1 does not result in other, "undesirable" derivations.

Next we can simulate the actions of A by applying rules from $P_{i,I}$ and P_i $(i=-1,0,1)$ to $\omega_{n,k}$. The position of the head of A is given by the position of the nonterminal q in the string. $\overline{P}_{0,I}P_0$ simulates an action of A with no head movement, $\overline{P}_{1,I}P_1$ takes care of a movement to the right A, and finally $\overline{P}_{-1,I}P_{-1}$ performs an action of A in which the head is moved to the left. At each moment of time there occurs at most one nonterminal q from Q in the sentential form. Therefore, reductions from $\overline{P}_{0,I}$, $\overline{P}_{1,I}$ and $\overline{P}_{-1,I}$ will always be applied to the correct substring. Note that these sets consist of fair reductions only. Due to these observations we have the following subderivations. There exist $c_i \in \overline{P}_{i,I}P_i$ $(i=-1, 0, 1)$ such that

- $p(a,D) \Rightarrow^{c_0}_{RO/f} q(a,E)$

for each $p,q \in Q$, $a \in \Sigma \cup \{\lambda\}$ and $D,E \in \Gamma$ such that $\delta(p,D) = (q,E, 0)$.

- $p(a,D) \Rightarrow^{c_1}_{RO/f} (a,E)q$

for each $p,q \in Q$, $a \in \Sigma \cup \{\lambda\}$ and $D,E \in \Gamma$ such that $\delta(p,D) = (q,E, 1)$.

- $(b,H)p(a,D) \Rightarrow^{c_{-1}}_{RO/f} q(b,H)(a,E)$

for each $p,q \in Q$, $a,b \in \Sigma \cup \{\lambda\}$ and $D,E,H \in \Gamma$ such that $\delta(p,D) = (q,E, -1)$.

Apart from these subderivations we also have that there exist control words d_0, e_0 in $\overline{P}_{0,I}P_0$ such that

$$p(a,D) \Rightarrow^{d_0}_{RO/f} p(a,D)$$

and

$$p(a,D) \Rightarrow^{e_0}_{RO/f} (p,a,D).$$

These latter two subderivations represent wrong guesses of the grammar (G,C) in the simulation of the Turing machine. However, they will not yield additional terminal strings. The first one for obvious reasons, and (p,a,D) can only be rewritten by one specific production from P_0. Analogous observations can be made with respect to $\overline{P}_{1,I}P_1$ and $\overline{P}_{-1,I}P_{-1}$. We can show by induction on the number of Turing machine moves that if

$$q_0 a_1 \dots a_n \vdash^*_A X_1 \dots X_{r-1} q X_r \dots X_{n+k},$$

then for some string c in $(\cup\{\overline{P}_{i,I} \cup P_i \mid i=-1,0,1\})^*$ we have

$$\omega_{n,k} \Rightarrow^{c}_{RO/f} \$(a_1,X_1)\dots(a_{r-1},X_{r-1})q(a_r,X_r)\dots(a_{n+k},X_{n+k}) \qquad (2)$$

where $a_i = \lambda$ $(i > n)$ and $X_i \in \Gamma$ $(1 \le i \le n+k)$. Let the derived string in (2) be

denoted by $\widetilde{X}_{r,q}^{n+k}$.

If a nonterminal symbol q from F appears in $\widetilde{X}_{r,q}^{n+k}$, then only rules from $\overline{P}_R$, $\overline{P}_L$ are applicable. Then it will be clear that there exists some control string d in $(P_R \cup P_L \cup \overline{P}_R \cup \overline{P}_L \cup P_\Sigma)^*$ such that

$$\widetilde{X}_{r,q}^{n+k} \Rightarrow_{RO/f}^{d} \$a_1 \ldots a_n q. \tag{3}$$

By applying a single rule from P_λ to this latter string we obtain the terminal string $\$a_1 \ldots a_n$.

Thus $\{\$\}T(A) \subseteq L_{RO/f}(G,C)$. The converse inclusion can be proved by induction in a similar way. Note that if $q_0 \in F$, then $L_{RO/f}(G,C) = \Sigma^*$.

For each Turing machine A we have constructed RCB/RO/S/f and RCB/RO/B/f grammars that generate $\{\$\}T(A)$. These grammars are trivially RCB/RO/S/g and RCB/RO/B/g grammars too, respectively. However, in these latter two cases we have to define C by $C = (P \cup \overline{P}_f)^*$, where $\overline{P}_f$ is the set of fair reductions induced by P. Note that this control language can be used for both the B and the S-mode; cf. the remark at the end of the construction of P.

Next we define a homomorphism $h : \Sigma \cup \{\$\} \to \Sigma^*$ by $h(\$) = \lambda$ and $h(a) = a$ for each a in Σ. Since the families of RCB/RO languages are closed under homomorphism (Proposition 3.3.b in Chapter II), we can effectively construct an RCB/RO grammar (G_0, C_0) such that

$$L_{RO}(G_0, C_0) = h(L_{RO}(G,C)) = h(\{\$\}T(A)) = h(\{\$\}L_0) = L_0.$$

This concludes the proof of the implication from right to left in 3.1. The converse implication can be proved using Church's Thesis. □

In the construction applied in the proof of Proposition 3.1 we defined the control language C equal to $(P \cup \overline{P})^*$ for the RO/B/f and the RO/S/f-mode. Thus we actually constructed an uncontrolled bidirectional grammar. Therefore, from the proof of Proposition 3.1 we obtain immediately the following consequence in which we use the concept of B grammar. A *bidirectional grammar* or *B grammar* is an RCB grammar (G,C) which satisfies $C = (P \cup \overline{P})^*$.

Corollary 3.2. *A language L_0 is recursively enumerable if and only if the language $\{\$\}L_0$ is a B/RO/f language.* □

From 3.1 and 3.2 it follows that providing B/RO/f grammars with control languages does not result in additional language generating power.

Both 3.1 and 3.2 are examples of characterizing the recursively enumerable languages in terms of rather simple means. We only use context-free rules but in both a productive and a reductive way. So the main results

of this section belong to a large class of similar characterizations of which [Asv86, BakBoo, Boa, Boo78, Cul, EngRoz, Sav] are a few instances only.

4. Time-Bounded λ-free RCB Grammars

For the definition of λ-free RCB or λRCB grammar we refer to Section III.3. The notions of bounding function, time-bounded λRCB grammar as well as the classes Φ_m, $POLY_m$, $POLY(k)_m$ with $k \geq 1$, and LIN_m (m is a mode) have been introduced in Section III.2. For each mode m and each bounding function ϕ, let $\mathbf{L}(\phi)$ denote the family of languages generated by λRCB/m grammars that are bounded by ϕ. Then we have that $\Phi_m = \cup\{\mathbf{L}_m(\phi) \mid \phi \in \Phi\}$.

Now we are ready to formulate a consequence of Proposition 3.1. It shows that for each mode m that includes the RO-submode, providing RCB/m grammars with a time bound is a real restriction in the sense that it results in a less powerful grammar model.

Corollary 4.1.
(1) *For each bounding function* ϕ, $\mathbf{L}_{RO}(\phi)$ *is a proper subfamily of* $\mathbf{L}_{RO}$.
(2) *For each family* Φ *of bounding functions,* $\Phi_{RO} \subset \mathbf{L}_{RO}$, *i.e.,* Φ_{RO} *is a proper subfamily of the family of RCB/RO languages.*

Proof. In Chapter III parsing algorithms for Φ_{RO} languages have been outlined. Since these algorithms terminate for each input, it follows that all languages in Φ_{RO} are recursive. □

Actually, we can slightly improve upon Corollary 4.1, for which we need the following concepts and notations. Let NTIME(ϕ) be the family of λ-free languages which are accepted by multi-tape nondeterministic Turing machines in time $\phi : \mathbb{N} \to \mathbb{N}$. As usual **NP** is defined by

$$\mathbf{NP} = \cup\{\text{NTIME}(n^d) \mid d \in \mathbb{N}\},$$

i.e., **NP** is the family of λ-free languages acceptable nondeterministically in polynomial time.

Proposition 4.2. (1) *Let* L_0 *be a* λ*RCB language bounded by a function* $\phi : \mathbb{N} \to \mathbb{N}$. *Then* $L_0 \in \text{NTIME}(\phi^2)$.
(2) $POLY \subseteq \mathbf{NP}$.

Proof (sketch). (1) Using a 2-tape nondeterministic Turing machine each rewriting step can be simulated in a constant number of steps. Looking for the prescribed substring to be rewritten requires an amount of time which does not exceed $c.\phi(n)$ for some $c \geq 1$, where n is the length of the string in L_0 to be accepted. Therefore, the total time is in $O(\phi^2)$. Note that this crude time bound holds for all different modes.
(2) This follows from (1) and the fact that the class of polynomials over $\mathbb{N}$

is closed under squaring. □

It remains an open question whether a kind of converse of Proposition 4.2(2) holds, i.e., whether there exists a mode m such that $\mathbf{NP} \subseteq POLY_m$. The problem in establishing such an inclusion is twofold. First, we have to simulate nondeterministic Turing machine computations by RCB grammars. This is the easy part since in the proof of Proposition 3.1 we can replace the control language $C = (P \cup \overline{P})^*$ by

$$\{\pi_1\}^*\{\pi_2\}P_{\Sigma\Sigma}^*\{\pi_3\}(\overline{P}_{\$R}P_{\$L})^*\{\overline{\pi}_4\pi_5\}(\overline{P}_{-1,I}P_{-1} \cup \overline{P}_{0,I}P_0 \cup \overline{P}_{1,I}P_1)^*(P_R \cup P_L \cup \overline{P}_R \cup \overline{P}_L \cup P_\Sigma)^*P_\lambda.$$

and simulate all nondeterministic transitions of A in a straightforward way. But the hard part is, of course, to do this simulation with a λ-free RCB grammar, since in general we have $k \neq 0$, i.e., A needs more than n tape cells for its computation. Therefore, some substantial amount of erasing seems to be inevitable. Probably, it is easier to show that NTIME$(n) \subseteq LIN$.

5. Modes of Derivation.

In this section we discuss some differences between the RO-mode and RA-mode and between the RN-mode and the RS-mode; cf. Sections I.3.2 and II.2.

When we apply a production to a sentential form with respect to the RO-mode, only one terminal can be rewritten. This is not reflected in the case of applying a reduction under RO-mode. In this latter case there is possibly more than one substring that can be rewritten. For example, in the string aBa the reduction $a \to A$ is applicable to both a's, i.e., we have $aBa \Rightarrow_{RO}^{a \to A} aBA$ and $aBa \Rightarrow_{RO}^{a \to A} ABa$.

The RA-mode has the property that when a rule is applicable to some sentential form, then precise one substring of this sentential form can be rewritten. In the example presented above, only the a on the right can be rewritten, i.e., $aBa \Rightarrow_{RA}^{a \to A} aBA$. There is another difference with the RO-mode, viz., $aBA \Rightarrow_{RA}^{a \to A} ABA$ does hold, in contrary to the RO-mode which does not permit this derivation. However, the generating power of the RA-mode is the same as the RO-mode.

Proposition 5.1. *A language L_0 is an RCB/RA language if and only if L_0 is recursively enumerable; i.e.,* $\mathbf{L}_{RA} = RE$.

Proof. In the construction in the proof of Proposition 3.1, the left-hand side of each reduction occurs at most once in each possible sentential form. Therefore the derivation according the RO-mode and the RA-mode will have

the same effect with respect to this particular grammar. □

In case of ordinary context-free grammars the RO-mode and RA-mode are also equivalent of course. Definition I.3.2.2 (RA-mode) uses the same condition as the RO-mode, but now this condition also applies to reductions as well. Another reason to prefer the RA-mode rather than the RO-mode shows up if we express both modes in the terminology of Thue systems. Consider P as a Thue system with alphabet V and the relation $\Leftrightarrow_P$ is defined as in Definition I.2.2.1. Then we have

$xuy \Rightarrow_{RA}^{u \rightarrow v} xvy$ if and only if

- $xuy \Leftrightarrow_P xvy$
- u occurs in uy only once
- if $u = \lambda$ then $y = \lambda$.

The RO-mode can be expressed in a similar way as follows.

$xuy \Rightarrow_{RO}^{u \rightarrow v} xvy$ if and only if

- $xuy \Leftrightarrow_P xvy$
- if u is in $V - \Sigma$

  then u does not occur in y

  else v does not occur in y.

Clearly, this is a less elegant property than in case of the RA-mode.

We can also use this description of the RO-mode and RA-mode both for the RN-mode and RS-mode. Let $\Sigma \subset V$. The RN-mode can be described by

$xuy \Rightarrow_{RN}^{u \rightarrow v} xvy$ if and only if

- $xuy \Leftrightarrow_P xvy$
- $y \in \Sigma^*$.

This allows us to write $Baa \Rightarrow_{RN/g}^{a \rightarrow A} BAa$, as well as $Baa \Rightarrow_{RN/g}^{a \rightarrow A} BaA$, where one would expect only the latter possibility. In the RS-mode at most one substring can be rewritten. In the terminology of Thue systems the RS-mode mode is characterized by

$xuy \Rightarrow_{RS}^{u \rightarrow v} xvy$ if and only if

- $xuy \Leftrightarrow_P xvy$
- u occurs in uy only once
- $y \in \Sigma^*$.
- if $u = \lambda$ then $y = \lambda$.

It is obvious that the following holds.

Proposition 5.2. *The modes RN and RS are equivalent when combined with the f-mode. Consequently,* $\mathbf{L}_{RS/f} = \mathbf{L}_{RN/f}$*, and for each family* Φ *of bounding functions we have* $\Phi_{RS/f} = \Phi_{RN/f}$.

Proof. Restricted to the f-mode, in the characterizations of both the RN- as the RS-mode the string u has to contain at least one nonterminal. □

So the families of RN/B/f and RN/S/f languages are equal to the families of RS/B/f and RS/S/f languages, respectively. For the other RN-modes it may be possible that the corresponding RS variant will result in a different language family. Viz., it might turn out that the family of RN/B/g and RN/S/g languages are not equal to the families of RS/B/g and RS/S/g languages, respectively. However, we observe that the properties of the families of RN/B/g and RN/S/g languages, established in Chapter II, also hold for the corresponding families of RS languages.

Proposition 5.3. *The families of RS/B/g and RS/S/g languages are closed under union and in particular under union with a regular set.* □

6. Concluding Remarks

We showed that the families of RCB/RO- and of RCB/RA languages coincide with the family of recursively enumerable languages (Propositions 3.1 and 5.2). Although it is not very difficult to simulate Turing machine computations by RCB/RO grammars we organized our construction in a way such that a minimum of control is sufficient; cf. Corollary 3.2. Our results are summarized in Table 1 in which *CFL* denotes the family of context-free languages. A question mark represents an open problem, viz. a language family that has not yet been characterized in terms of a well-known member of the extended Chomsky-hierarchy. These "unknown" language families properly include *CFL* (Proposition II.2.4) and are, of course, included in *RE*.

m	RO				RN			
	B		S		B		S	
	f	g	f	g	f	g	f	g
$\mathbf{L}_m$	*RE*	*RE*	*RE*	*RE*	*CFL*	?	?	?

Table 1.

Whether there exist similar characterizations for the complexity classes

- **NP** in terms of polynomial time-bounded RCB/RO grammars, and
- NTIME(n) in terms of linear time-bounded RCB/RO grammars

remains open; cf Section 4. Another open problem is the question whether there exists a natural restriction on RCB/m grammars that characterizes NSPACE(n), i.e., the family of languages acceptable nondeterministically by Turing machines in linear space.

CHAPTER V

Regularly Controlled Bidirectional Extended Linear Basic Grammars

1. Introduction

In this chapter we study the concept of bidirectional application of productions – i.e., using a production of a grammar as a reduction too – with respect to regularly controlled extended linear basic (macro) grammars [AsvEng79]. The resulting new grammatical model is in essence equal to the regularly controlled bidirectional context-free grammars of Chapter II in which the underlying context-free grammar has been replaced by an extended linear basic grammar. We motivate the choice of (K-)extended linear basic grammars by the fact that for some language families K the family of languages $LB_m(K)$ is incomparable with CFL. This holds for instance if K equals $\varnothing NE$ or FIN and $m = \mathrm{IO}$, since in these cases we have $LB = LB_{\mathrm{IO}}(K)$.

The structure of this chapter is as follows. In Section 2 we introduce regularly controlled bidirectional (m,K)-elb grammars or (m,REG,K)-belb grammars. Then we formally define for these (m,REG,K)-belb grammars the RS/B/f-mode of derivation. The corresponding grammars are called (r,f,m,REG,K)-belb grammars. In addition, we introduce c-trees to visualize the structure of sentential forms generated by (r,f,m,REG,K)-belb grammars. These c-trees are also helpful in the proofs of Section 5. Closure properties of the family $RBLB_{r,f,m}(K)$ of (r,f,m,REG,K)-belb languages are established in Section 3. For both modes OI and IO and under weak assumptions on the family K it is shown that the family $RBLB_{r,f,m}(K)$ is closed under the regular operations (union, concatenation, and Kleene +). Furthermore, we will prove that if K is a nontrivial family of languages closed under ngsm mappings, then $RBLB_{r,f,\mathrm{OI}}(K)$ is a full substitution-closed AFL. Concerning the family $RBLB_{r,f,\mathrm{IO}}(K)$, we establish – under appropriate conditions on K – closure under intersection with regular languages and deterministic substitution; hence this family is a full QAFL (in the sense of [AsvEng79]) closed under deterministic substitution. In

Section 4 we discuss the language generating capacity of (r,f,m,REG,K)-belb grammars. We show that the family $RBLB_{r,f,\mathrm{OI}}(\varnothing NE)$ is equal to the family *OI* of OI-macro languages and that the family *IO* of IO-macro languages is included in the family $RBLB_{r,f,\mathrm{IO}}(\varnothing NE)$. In Section 5 (m,REG,K)-belb grammars provided with free application of rules are studied. However, the restriction of allowing only fair reductions is maintained. Then for $m = \mathrm{OI}$ and for $m = \mathrm{IO}$ the family of languages generated by these so-called (f,m,REG,K)-belb grammars equals the family of recursively enumerable languages. Finally, Section 6 contains some concluding remarks.

2. Regularly Controlled Bidirectional (m,K)-elb Grammars

First, we note that in Definition I.3.3.1(i) we required that $k \geq 1$, whereas in the original definition of (m,K)-elb grammars in [AsvEng79] $k = 0$ is also permitted. However, the restriction to $k \geq 1$ causes no loss of generality, except that in our approach we need that the family K contains at least one nonempty language. This will be proved in Lemma 2.2.

Definition 2.1. An (m,K)-elb grammar $G = (\Phi,\Psi,\Sigma,X,P,S)$, is in *equal length form* if there is a natural number n $(n \geq 0)$ such that each nonterminal in Φ is either in Φ_0 and equal to S, or in Φ_n and each language name in Ψ is either in Ψ_0 or in Ψ_n. □

Lemma 2.2. *Let K be a language family that contains a nonempty language L_0. For each (m,K)-elb grammar G_0 there exists an (m,K)-belb grammar G in equal length form such that $L_m(G) = L_m(G_0)$.*

Proof (sketch). Let $G_0 = (\Phi,\Psi,\Sigma,X,P,S)$ be an (m,K)-elb grammar with $X_0 = \{x_1,\ldots,x_n\}$. We enlarge the rank of each nonterminal unequal to S in Φ_k, where $k \geq 0$, and of each language name in Ψ_k, where $k \geq 0$, to n by adding $n-k$ dummy arguments. To the resulting alphabet Ψ' we add each language name with zero rank which occurs in an initial production of P. The productions are changed accordingly by introducing two new language names ψ_e in Ψ_n and ψ_z in Ψ_0, with $\psi_e(\vec{x}) \to L_0$ and $\psi_z \to L_0$.

This well-known construction – e.g., cf. [Asv78] – can easily be written out in full detail. □

Now we introduce regularly controlled bidirectional (m,K)-elb grammars. They consist of an (m,K)-elb grammar provided with a regular control language over $P \cup \bar{P}$. We define the set of reductions $\bar{P}$ corresponding to P as in Section I.3.3. Furthermore, for each production π we define $\bar{\bar{\pi}}$ to be equal to π. An element of $P \cup \bar{P}$ will be called a *rule* as in previous chapters.

Notice that (m,K)-elb grammars provided with an arbitrary control language over P have been studied in [Asv78, AsvEng79].

Definition 2.3. A *regularly controlled bidirectional* (m,K)*-elb grammar* or (m,REG,K)*-belb grammar* is a triple $(G,C,\not\subset)$ where

- G is an (m,K)-elb grammar (Φ,Ψ,Σ,X,P,S),
- C is a regular language with $C \subseteq (P \cup \overline{P})^*$,
- $\not\subset$ is a special symbol not occurring in Φ, Ψ, Σ or X.

We call G the *underlying grammar* of $(G,C,\not\subset)$ and C is called the *control language* of $(G,C,\not\subset)$. Sentences of C will be referred to as *control words*. □

Definition 2.4. A production $A(x_1,\ldots,x_n) \to t$ of a macro grammar is called *argument preserving* [Fis68a] if each variable x_i $(1 \le i \le n)$ occurs in the term t.

Let $G = (\Phi,\Psi,\Sigma,X,P,S)$ be an (m,K)-elb grammar. A production of the form $\psi(x_1,\ldots,x_n) \to L_0$ in P, where $\psi \in \Psi_n$, is called *argument preserving* if each variable x_i $(1 \le i \le n)$ occurs in each word w from L_0. □

Note that productions of the form I.3.3.1(i) and I.3.3.1(ii) are argument preserving by definition.

For an (m,REG,K)-belb grammar $(G,C,\not\subset)$, with $G = (\Phi,\Psi,\Sigma,X,P,S)$, let and $Term(G,\not\subset)$ denote the set of terms $T(\Sigma \cup X \cup \Phi \cup \Psi \cup \{\not\subset\})$. With each (m,REG,K)-belb grammar we associate – as usual – a derivation relation. This derivation relation formalizes bidirectional right-most rewriting; cf. Definition 2.6.

Definition 2.5. Let $(G,C,\not\subset)$ be an (m,REG,K)-belb grammar, where $G = (\Phi,\Psi,\Sigma,X,P,S)$. Let ρ be a rule from $P \cup \overline{P}$, where $\alpha[\vec{x}]$ is the left-hand side of ρ, and let τ be a term in $Term(G,\not\subset)$.

(a) If $\alpha[\vec{x}]$ is of the form $Z(\ldots)$, with $Z \in \Phi \cup \Psi$, then we say that τ *fits in with* ρ in case there are arguments $t_1,\ldots,t_n$ from $Term(G,\not\subset)$ such that $\tau = \alpha[t_1,\ldots,t_n]$, where $\alpha[t_1,\ldots,t_n]$ is the result of substituting the terms $t_1,\ldots,t_n$ for $x_1,\ldots,x_n$ in $\alpha[\vec{x}]$, respectively.

(b) If $\alpha[\vec{x}]$ is a language $L_0 \subseteq (\Sigma \cup X)^*$, i.e., ρ is a reduction of the form $L_0 \to \psi(\vec{x})$, then τ *fits in with* ρ if there is at least one string t in L_0 such that $\tau = t[t_1,\ldots,t_n]$, where $t[t_1,\ldots,t_n]$ is the result of substituting the terms $t_1,\ldots,t_n$ for $x_1,\ldots,x_n$ in t, respectively. □

Definition 2.6. Let $(G,C,\not\subset)$ be an (m,REG,K)-belb grammar, where $G = (\Phi,\Psi,\Sigma,X,P,S)$. Let ρ be rule from $P \cup \overline{P}$, and σ, τ be terms in $Term(G,\not\subset)$. We write $\sigma \Rightarrow^{\rho}_{r,m} \tau$ if there exists a term u in $Term(G,\not\subset)$, and strings v, x and y over the alphabet $\Phi \cup \Psi \cup \Sigma \cup X \cup PC$ such that $\sigma = xuy$ and $\tau = xvy$ and

- y contains no symbol from $\Phi \cup \Psi$,
- if λ fits in with ρ then $u = y = \lambda$,
- u is the only subterm in uy that fits in with ρ,
- either ρ is a production, τ is the result of rewriting σ by ρ, and $\sigma \Rightarrow_m \tau$, or ρ is a reduction, σ is the result of rewriting τ by $\bar{\rho}$, and $\tau \Rightarrow_m \sigma$. □

Let $C \subseteq (P \cup \bar{P})^*$ be a control language. A control word c in C is a sequence of rules. The application of a sequence of rules from $P \cup \bar{P}$ to a term τ is defined as the successive application of the rules which constitute c. Viz., if $c = \rho_1 \ldots \rho_n$ $(n \geq 0)$, then we write $\tau \Rightarrow^c_{r,m} \tau'$ if there are terms τ_i $(0 \leq i \leq n)$ such that $\tau_0 = \tau$, $\tau_n = \tau'$ and for each i $(0 \leq i < n)$ $\tau_i \Rightarrow^{\rho_i}_{r,m} \tau_{i+1}$ holds. In case a rule ρ_i in c is not applicable to the term τ_i, then further application of rules is blocked, and the application of c to τ yields no result, i.e., there is no term τ' such that $\tau \Rightarrow^c_{r,m} \tau'$ is defined.

Definition 2.7. An (m,REG,K)-belb grammar provided with right-most rewriting will be called an (r,m,REG,K)-*belb grammar*. Let $(G,C,\text{¢})$ be an (r,m,REG,K)-belb grammar with underlying grammar $G = (\Phi,\Psi,\Sigma,X,P,S)$ and control language $C \subseteq (P \cup \bar{P})^*$. Then the *language* generated by $(G,C,\text{¢})$ under the mode r,m is defined by

$$L_{r,m}(G,C,\text{¢}) = \{w \in \Sigma^* \mid \exists\, c \in C \,.\, S \Rightarrow^c_{r,m} w\}.$$

The *family* of languages generated by (r,m,REG,K)-belb grammars is denoted by $RBLB_{r,m}(K)$. □

The derivation relation $\Rightarrow_{r,m}$ defined above corresponds to the RS/B-mode of derivation as defined in Chapter I for RCB grammars.

Example 2.8. Let L_1 be the language of equal length substrings, i.e.,

$$L_1 = \{x_1 c x_2 \ldots c x_n \mid x_i \in \{a,b\}^*,\ |x_i| = m,\ 1 \leq i \leq n,\ n,m \geq 1\}.$$

In [Fis68a] it is shown that this language can be generated by an OI macro grammar. The language L_1 belongs to $RBLB_{r,\mathrm{OI}}(\varnothing NE)$, i.e., it can also be generated by the following $(r,\mathrm{OI},REG,\varnothing NE)$-belb grammar $(G,C,\text{¢})$.

Define G by (Φ,Ψ,Σ,X,P,S), where the set of nonterminals Φ is equal to $\{S,A,B,D,E,F,H\}$, and S is the start symbol. The alphabet of language names Ψ equals $\{\psi_i \mid 0 \leq i \leq 9\}$. The set Σ of terminals is $\{a,b,c\}$, and $X = \{x,y\}$. Finally, the set of productions P of G consists of

$$\begin{aligned}
&\pi_0 = S \to A(\psi_0), && \pi_{12} = F \to \psi_6,\\
&\pi_1 = A(x) \to A(\psi_1(x)), && \pi_{13} = F \to \psi_7,\\
&\pi_2 = A(x) \to B(\psi_2(x)), && \pi_{14} = \psi_0 \to \varnothing,
\end{aligned}$$

$\pi_3 = B(x) \to \psi_3(x)$, $\pi_{15} = \psi_6 \to \{a\}$,
$\pi_4 = A(x) \to \psi_2(x)$, $\pi_{16} = \psi_7 \to \{b\}$,
$\pi_5 = D(x) \to \psi_1(x)$, $\pi_{17} = H(x) \to \psi_4(x)$,
$\pi_6 = D(x) \to E(\psi_2(x), \psi_4(x))$, $\pi_{18} = H(x) \to \psi_8(x)$,
$\pi_7 = E(x,y) \to \psi_5(x,y)$, $\pi_{19} = H(x) \to \psi_9(x)$,
$\pi_8 = \psi_5(x,y) \to \{xy\}$, $\pi_{20} = \psi_4(x) \to \varnothing$,
$\pi_9 = \psi_3(x) \to \{xcx\}$, $\pi_{21} = \psi_8(x) \to \{a\}$,
$\pi_{10} = \psi_2(x) \to \{x\}$, $\pi_{22} = \psi_9(x) \to \{b\}$.
$\pi_{11} = F \to \psi_0$,

The rank of the symbols in $\Phi \cup \Psi$ are easy to infer from the productions in P. Define the control language C by the regular expression

$$\pi_0\pi_1^*(\pi_2\pi_3\pi_9\overline{\pi}_4)^*\pi_4(\pi_{10}(\overline{\pi}_5\pi_6\pi_7\pi_8\overline{\pi}_{17}(\pi_{18}\pi_{21}+\pi_{19}\pi_{22})+\overline{\pi}_{11}(\pi_{12}\pi_{15}+\pi_{13}\pi_{16})))^+.$$

First, a string $A(\psi_1(\ldots\psi_1(\psi_0)\ldots)$, in which the language name ψ_1 occurs n times, is generated by $\pi_0\pi_1^n$ $(n \geq 0)$. We represent the argument of A by $[n]$, where [0] represents ψ_0. By applying k times $(k \geq 0)$ $\pi_2\pi_3\pi_9\overline{\pi}_4$, followed by π_4 we obtain the string $\psi_2([n])c\ldots c\,\psi_2([n])$, which contains k times the symbol c. In the following we discuss the expanding of a substring $\psi_2([n])$. By π_{10} we derive $\psi_2([n])$ to $[n]$, which is rewritten to $\psi_2([n-1])H([n-1])$ by the subsequence $\overline{\pi}_5\pi_6\pi_7\pi_8\overline{\pi}_{17}$ in case $n \geq 1$, and to F by the reduction $\overline{\pi}_{11}$ in case $n = 0$. Next, both $H([n-1])$ and F are expanded to a or b by the sequences $\pi_{18}\pi_{21}+\pi_{19}\pi_{22}$ and $\pi_{12}\pi_{15}+\pi_{13}\pi_{16}$, respectively.

Then $L_{r,\mathrm{OI}}(G,C,¢) = L_1$, which can now be easily verified. □

Although it is straightforward to define reductions associated with productions of the form I.3.3.1(iii) (cf. Definition 2.6.), we do not study grammatical models in which such (arbitrary) reductions occur. These terminal reductions have the effect that they allow terminals to act as some kind of nonterminal symbol, which makes the distinction between terminals and nonterminals unclear. We have noticed this problem already in the case of regularly controlled bidirectional grammars that have a context-free grammar as its underlying grammar; cf. Chapter II. This restriction means that in this chapter we only study the *fair mode* – cf. Chapter II – of bidirectional rewriting, in which we disallow terminal reductions. As a consequence, it enables us to omit the symbol ¢. We will call this type of grammar an (r,f,m,REG,K)-*belb grammar*. The family of languages generated by (r,f,m,REG,K)-belb grammars is denoted by $RBLB_{r,f,m}(K)$. Note that the language of Example 2.8 can be generated by an $(r,f,\mathrm{OI},REG,\varnothing NE)$-belb

grammar.

In the remainder of this section we clarify the structure of sentential forms generated by (r,f,m,REG,K)-belb grammars. Since the family of regular languages is closed under intersection, we can put regular restrictions on the control language. In the sequel, we assume that the set of productions P is the union of the disjoint sets P_1, P_2 and P_3, where P_1, P_2 and P_3 consist of productions of the form I.3.3.1(i), I.3.3.1(ii) and I.3.3.1(iii), respectively. Then we assume without loss of generality that for an (r,f,OI,REG,K)-belb grammar (G,C) with underlying grammar $G=(\Phi,\Psi,\Sigma,X,P,S)$, the control language C is included in

$$((P_1\bar{P}_1\cup P_2\bar{P}_2)^*(P_1\cup\bar{P}_1\cup P_2P_3^+(\bar{P}_2\cup\{\lambda\})))^+. \tag{1}$$

For the same reason we can assume that an (r,f,IO,REG,K)-belb grammar (G,C) possesses a control language C which is included in

$$((\bar{P}_2P_2\cup P_3)^*(\bar{P}_2\cup\{\lambda\})(P_1(\bar{P}_2P_2)^*\bar{P}_1\cup P_2\bar{P}_2)^*(P_1\cup P_2P_3))^+. \tag{2}$$

In addition, for both modes OI and IO we may assume that the first rule of each control word in C is an initial production.

The restriction to control languages which are included in (1) or (2) becomes apparent when we inspect the structure of a sentential form occurring in the derivation according to an (r,f,m,REG,K)-belb grammar (G,C). We represent terms from $Term(G)$ as follows. Define a *c-tree* as a variation on the well-known tree structure in which now the nodes are strings over $\Phi\cup\Psi\cup\Sigma\cup X\cup\{\#\}$. The symbol # is used to denote the concatenation operation in $T(\Phi\cup\Psi\cup\Sigma\cup X)$ explicitly. If A is an element of $\Phi\cup\Psi$ with rank n and $n\geq 1$, then A has n descendants which are again c-trees. If a node α is a string of symbols of rank zero, i.e., $\alpha\in(\Phi_0\cup\Psi_0\cup\Sigma\cup X\cup\{\#\})^*$, then α is called a *leave*. As an example, the term

$$A(\psi(x_1,ax_2b)x_1,ab)\psi(a,b)x_1$$

is represented by the c-tree in Figure 1.

Note that a c-tree does not represent a derivation of the grammar (G,C).

A derivation corresponding to an (r,f,OI,REG,K)-belb grammar consists of a sequence of sentential forms which have the form

$$\psi_1(\vec{\psi}_1(\vec{t}_1))w_1\ldots w_{n-1}\psi_n(\vec{\psi}_n(\vec{t}_n))w_nA(\vec{\psi}_0(\vec{t}_0))w_0. \tag{3}$$

The formula $\vec{\psi}_i(\vec{t}_i)$ is the abbreviation of

$$\psi_{i1}(t_{i1}),\ldots,\psi_{ir(i)}(t_{ir(i)}) \quad \text{with} \quad 0\leq i\leq n.$$

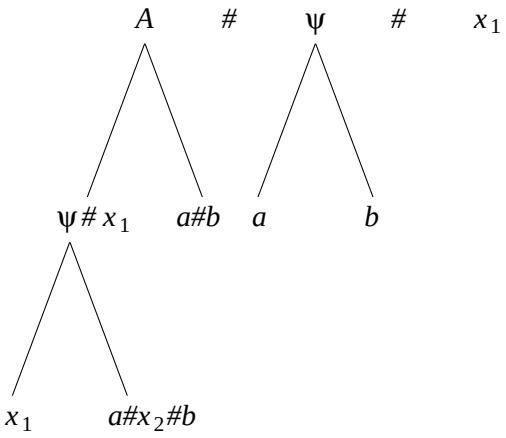

Figure 1.

In (3) the following notational conventions are used. The symbols ψ_{ij} are language names. The number of language names may be zero, i.e., $n \geq 0$. The symbol A is either a language name or a nonterminal. A terminal string is a possible sentential form, so the substring $A(\ldots)$ is optional in (3). Each t_{ij} is a list of terms over Ψ, each of which can be represented by a c-tree in which each node is a language name; the leaves of each term in t_{ij} are language names of rank zero. In the c-tree representation of (3), shown in Figure 2, these t_{ij}'s are represented by a triangle. The terms w_i are strings in $(\Sigma \cup X)^*$. Note that $r(i)$ is equal to the rank of ψ_i $(i \geq 1)$, and $r(0)$ is equal to the rank of A.

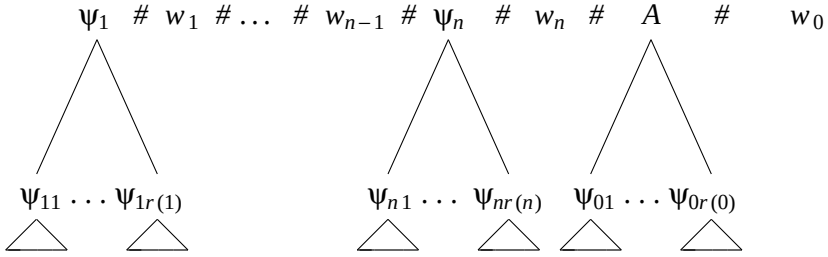

Figure 2.

A sentential form generated by an $(r, f, \mathrm{IO}, REG, K)$-belb grammar (G, C) is of the form

$$A_0(\psi_{01}(t_{01}),\ldots,\psi_{0i_0}(t_{0i_0}),A_1(\psi_{11}(t_{11}),\ldots,\psi_{1i_1}(t_{1i_1}),A_2(\ldots,A_n(\psi_{n1}(t_{n1}),\ldots,$$
$$\psi_{ni_n}(t_{ni_n}),w_{ni_n+1},\ldots,w_{r(n)}),\ldots,w_{r(2)}),w_{1i_1+2},\ldots,w_{r(1)}),w_{0i_0+2},\ldots,w_{r(0)}). \quad (4)$$

In this sentential form (4) the A_i's are nonterminals, the ψ_{ij}'s denote language names, the w_{ij}'s are strings over $\Sigma \cup X$, and each t_{pq} denotes a list of w_{ij}'s the length of which is equal to the rank of ψ_{pq}. Furthermore, $r(i)$ $(0 \le i \le n)$ is equal to the rank of A_i. The sentential form (4) is represented by a c-tree in Figure 3 in which each t_{ij} is represented by a triangle.

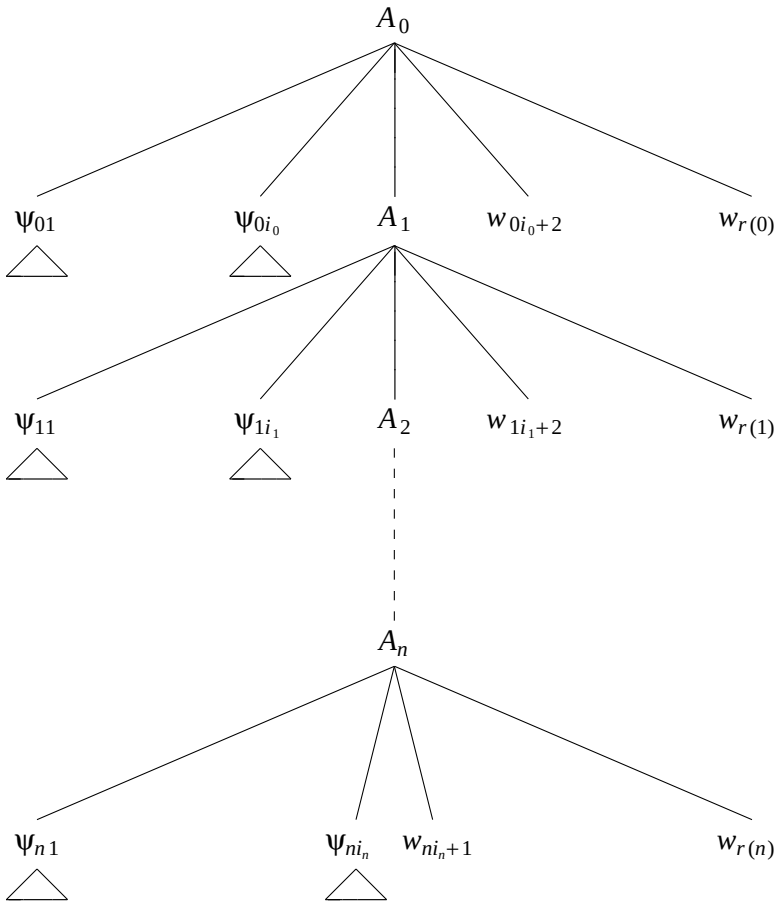

Figure 3.

3. Properties of $RBLB_{r,f,m}(K)$-languages

In the proofs of the following propositions we assume that L_i $(i \ge 1)$ is a language generated by an (r, f, m, REG, K)-belb grammar (G_i, C_i) with $G_i = (\Phi_{(i)}, \Psi_{(i)}, \Sigma_i, X_i, P_{(i)}, S_i)$. Thus $\Phi_{(i)}$ and $\Psi_{(i)}$ are ranked alphabets, i.e., $\Phi_{(i)} = \cup \Phi_{(i)j}$ and $\Psi_{(i)} = \cup \Psi_{(i)j}$. We assume that the sets of language names and the sets of variables of these grammars are mutually disjoint, i.e., $i \ne j$ implies $\Phi_{(i)} \cap \Phi_{(j)} = \emptyset$, $\Psi_{(i)} \cap \Psi_{(j)} = \emptyset$ and $X_i \cap X_j = \emptyset$.

Remember that a family of languages K is closed under *left-* [*right-*] *marking* if for each language L_0 in K, the language $\{\$\}L_0$ [$L_0\{\$\}$, respectively] is in K, where the symbol \$ does not occur in the alphabet of L_0. Frequently, we write $\$L_0$ instead of $\{\$\}L_0$; cf. Section II.6.

The family *SYMBOL* is defined as the family of all languages consisting of a single word which is of length one, i.e., $SYMBOL = \{\{a\} \mid a \in \Sigma_\omega\}$ where Σ_ω is a countably infinite set of terminal symbols.

Proposition 3.1. *Let K be a family of languages closed under left- or right-marking. If $K \supseteq SYMBOL$, then $RBLB_{r,f,m}(K)$ is closed under union, concatenation, Kleene + and Kleene $*$.*

Proof. Union. Straightforward. This even holds without the premisses on the family K.

Concatenation. We construct an (r,f,m,REG,K)-belb grammar (G,C) from (G_1,C_1) and (G_2,C_2) such that $L_{r,m}(G,C) = L_1L_2$. For both modes OI and IO we can use the same underlying grammar G. Define G equal to $(\Phi,\Psi,\Sigma_1,X_1,P,S)$, where $\Phi = \Phi_{(1)} \cup \Phi_{(2)} \cup \{S,Z\}$, $S \in \Phi_0$, $Z \in \Phi_2$, and where $\Psi = \Psi_{(1)} \cup \Psi_{(2)} \cup \{\psi,\psi_1,\psi_2\}$ such that $\psi_1,\psi_2 \in \Psi_0$ and $\psi \in \Psi_2$. Furthermore, we assume that S, Z, ψ, ψ_1, and ψ_2 are new symbols. The set of productions P is equal to $P_{(1)} \cup P_{(2)} \cup \{\pi_\psi,\pi_Z,\pi_0,\pi_1,\pi_2\}$ with $\pi_\psi = \psi(x,y) \to \{xy\}$, $\pi_Z = Z(x,y) \to \psi(x,y)$, $\pi_0 = S \to Z(\psi_1,\psi_2)$, $\pi_1 = S_1 \to \psi_1$, and $\pi_2 = S_2 \to \psi_2$. Note that $\{xy\}$ belongs to K, as K includes the family *SYMBOL*, $x \neq y$, and K is closed under left- or right-marking. Now if $m = $ OI, then define the control language C by $\pi_0\pi_Z\pi_\psi\bar{\pi}_2C_2\bar{\pi}_1C_1$. Otherwise, if $m = $ IO, then we define C by $\pi_0\bar{\pi}_2C_2\bar{\pi}_1C_1\pi_Z\pi_\psi$.

Kleene +. As in the case of concatenation, we construct an (r,f,m,REG,K)-belb grammar (G,C) from (G_1,C_1), that generates L_1^+. The underlying grammar G is for both modes OI and IO the same, but the control languages are different. Viz., define G by (Φ,Ψ,Σ,X,P,S), where $\Phi = \Phi_{(1)} \cup \{S,Z\}$ with $S \in \Phi_0$ and $Z \in \Phi_2$, and where the set Ψ is equal to $\Psi_{(1)} \cup \{\psi,\psi_1,\psi_2\}$, with $\psi_1,\psi_2 \in \Psi_0$ and $\psi \in \Psi_2$. Again S, Z, ψ, ψ_1, and ψ_2 are new symbols. We also assume $\Phi_{(1)} \cap \{S,Z\} = \varnothing$ and $\Psi_{(1)} \cap \{\psi,\psi_1,\psi_2\} = \varnothing$. The set of productions P is formed by $P_{(1)} \cup \{\pi_\psi,\pi_Z,\pi_S,\pi_0,\pi_1\}$, where $\pi_S = S \to S_1$, $\pi_\psi = \psi(x,y) \to \{xy\}$, $\pi_Z = Z(x,y) \to \psi(x,y)$, $\pi_0 = S_1 \to Z(\underline{\psi}_1,\underline{\psi}_1)$, and $\pi_1 = S_1 \to \psi_1$. The control language C is equal to $\pi_S(\pi_0\pi_Z\pi_\psi\bar{\pi}_1C_1\underline{\pi}_1)^*C_1$ if m is equal to OI, and in case of the IO-mode we take C equal to $\pi_S(\pi_0\bar{\pi}_1C_1\underline{\pi}_1)^*C_1(\pi_Z\pi_\psi)^*$.

Kleene $*$. Straightforward.

Note that in the proofs presented above the control languages have for both modes OI and IO a form in accordance with (1) and (2) from Section 2,

respectively. □

In the next proposition we show the closure under ngsm mappings of the language family $RBLB_{r,f,\mathrm{OI}}(K)$. Therefore, we recall the following definition.

Definition 3.2. An *ngsm* or a *nondeterministic generalized sequential machine* is a 6-tuple $T = (Q, \Sigma, \Delta, \delta, q_0, Q_F)$, where

- Q is a finite alphabet of *states*,
- Σ is a finite alphabet of *input symbols*,
- Δ is a finite alphabet of *output symbols*,
- $q_0 \in Q$ is the *initial state*,
- $Q_F \subseteq Q$ is the set of *accepting states*,
- δ is a mapping from $Q \times \Sigma$ into the finite subsets of $Q \times \Delta^*$.

As usual, δ is extended to a function from $Q \times \Sigma^*$ into the finite subsets of $Q \times \Delta^*$ as follows.

(i) $\delta(q, \lambda) = \{(q, \lambda)\}$,

(ii) For $q \in Q$, $x \in \Sigma^*$ and $a \in \Sigma$,

$$\delta(q,xa) = \{(p,w) \mid w = w_1 w_2 \text{ and for some } r \text{ in } Q,\ (r,w_1) \text{ is in } \delta(q,x) \text{ and } (p,w_2) \text{ is in } \delta(r,a)\}.$$

The mapping associated with $T = (Q, \Sigma, \Delta, \delta, q_0, Q_F)$ – called an *ngsm mapping* and denoted by T too – is the function $T: \Sigma^* \to 2^{\Delta^*}$ defined by $T(w) = \{z \mid (q,z) \in \delta(q_0,w),\ q \in Q_F\}$. The extension of T to a language L_0 over Σ is defined by $T(L_0) = \cup\{T(w) \mid w \in L_0\}$. □

The proof of the following proposition is performed by applying the well-known ‘‘triple’’ construction.

Proposition 3.3. *Let K be a family closed under ngsm mappings. Then $RBLB_{r,f,\mathrm{OI}}(K)$ is closed under ngsm mappings.*

Proof. Let (G_1, C_1) be an $(r, f, \mathrm{OI}, REG, K)$-belb grammar with $G_1 = (\Phi_{(1)}, \Psi_{(1)}, \Sigma, X_1, P_{(1)}, S_1)$, $C_1 \subseteq (P_{(1)} \cup \bar{P}_{(1)})^*$, and let T be an ngsm with $T = (Q, \Sigma, \Delta, \delta, q_0, Q_F)$. We construct an $(r, f, \mathrm{OI}, REG, K)$-belb grammar (G, C) such that $L_{r,\mathrm{OI}}(G, C) = T(L_{r,\mathrm{OI}}(G_1, C_1))$. Define the set of variables X as $\{x_{i[p,q]} \mid 1 \le i \le |X_1|,\ p,q \in Q\}$, and let $Q = \{q_0, \ldots, q_N\}$. With the list $x_1, \ldots, x_n$ we associate the list $x_{1[q_0,q_0]}, \ldots, x_{n[q_N,q_N]}$, denoted by $\tilde{x}$. It consists of $n\,|Q|^2$ different variables from X; $|Q|$ is the cardinality of the set Q. Let $\tilde{\psi}(\tilde{x})$ denote

$$(q_0\ \psi\ q_0)(\tilde{x}), \ldots, (q_N\ \psi\ q_N)(\tilde{x}).$$

For each $p,q \in Q$, let T_{pq} be the ngsm mapping induced by the ngsm $(Q, \Sigma \cup X_1, \Sigma \cup X, \delta', p, \{q\})$, where $\delta' : Q \times (\Sigma \cup X_1) \to 2^{(Q \times (\Sigma \cup X))^*}$ is the mapping defined by

$$\delta'(s,y) = \{(t,z) \mid (t,z) \in \delta(s,y),\ y \in \Sigma\} \cup \{(t, x_{i\,[s,t]}) \mid t \in Q,\ x_i = y,\ y \in X_1\}.$$

Furthermore, let U be the union of the sets

$$\{(p\ A\ q)(\tilde{x}) \to (p\ B\ q)(\tilde{\psi}_1(\tilde{x}), \ldots, \tilde{\psi}_k(\tilde{x})) \mid A \in \Phi_n,\ B \in \Phi_k,\ p,q \in Q,\ \psi_i \in \Psi_n,\ n \geq 0,\ k \geq 1\},$$

$$\{(p\ A\ q)(\tilde{x}) \to (p\ \psi\ q)(\tilde{x}) \mid A \in \Phi_n, \psi \in \Psi_n,\ n \geq 0,\ p,q \in Q\},\ \text{and}$$

$$\{(p\ \psi\ q)(\tilde{x}) \to T_{pq}(L_0) \mid \psi \in \Psi_n,\ p,q \in Q,\ \psi(\vec{x}) \to L_0 \in P_{(1)}\}.$$

Define a finite substitution $\tau : (P_{(1)} \cup (\overline{P_{(1)} - P_{(1)3}}))^* \to 2^{(U \cup \bar{U})^*}$ by

$$\tau(S_1 \to A(\psi_1, \ldots, \psi_n)) = \{(q_0\ S_1\ q_f) \to (q_0\ A\ q_f)(\tilde{\psi}_1, \ldots, \tilde{\psi}_n) \mid q_f \in Q_F\},$$

$$\tau(A(\vec{x}) \to B(\psi_1(\vec{x}), \ldots, \psi_k(\vec{x}))) = \{(p\ A\ q)(\tilde{x}) \to (p\ B\ q)(\tilde{\psi}_1(\tilde{x}), \ldots, \tilde{\psi}_k(\tilde{x})) \mid p,q \in Q\},$$

$$\tau(A(\vec{x}) \to \psi(\vec{x})) = \{(p\ A\ q)(\tilde{x}) \to (p\ \psi\ q)(\tilde{x}) \mid p,q \in Q\},$$

$$\tau(\psi(\vec{x}) \to L_0) = \{(p\ \psi\ q)(\tilde{x}) \to T_{pq}(L_0) \mid p,q \in Q\}.$$

Since there are no terminal reductions involved, a reduction $\bar{\pi}$ is always a reduction associated with an argument-preserving production of type I.3.3.1(i) or I.3.3.1(ii). Therefore, we can define $\tau(\bar{\pi})$ equal to $\overline{\tau(\pi)}$.

Then we define G equal to $(\Phi, \Psi, \Sigma, X, P, S)$, where Φ is given by

$\Phi_0 = \{S\} \cup \{(p\ A\ q) \mid A \in \Phi_{(1)0},\ p,q \in Q\}$,

$\Phi_{n|Q|^2} = \{(p\ A\ q) \mid A \in \Phi_{(1)n},\ p,q \in Q\}$ for each n $(n \geq 1)$,

$\Phi_l = \varnothing$ if there is no $n \in \mathbb{N}$ with $l = n\,|Q|^2$,

and Ψ is given by

$\Psi_{n|Q|^2} = \{(p\ \psi\ q) \mid \psi \in \Psi_{(1)n},\ p,q \in Q\}$ for each n $(n \geq 0)$,

$\Psi_l = \varnothing$ if there is no $n \in \mathbb{N}$ with $l = n\,|Q|^2$.

The set of productions P equals $\tau(P_{(1)}) \cup P_S$, where P_S is equal to $\{S \to (q_0\ S_1\ q_f) \mid q_f \in Q_F\}$, and, finally, we define the new control language C equal to $P_S \tau(C_1)$. Then $L_{r,\mathrm{OI}}(G,C) = T(L_{r,\mathrm{OI}}(G_1, C_1))$. □

Remember that a family closed under ngsm mappings if and only if it is closed under intersection with regular languages and under finite substitution; cf. Lemma 9.3 in [HopUll69]. Therefore, a direct consequence of Proposition 3.3 is the following result.

Corollary 3.4. *Let K be a family closed under ngsm mappings. Then the family of languages $RBLB_{r,f,\mathrm{OI}}(K)$ is closed under intersection with regular languages and under finite substitution.* □

Next we establish closure under two types of substitution. First we give precise definitions of substituting words for symbols in a word "nondeterministically" (Definition 3.5) and "deterministically" (Definition 3.6).

Definition 3.5. Let K be a family of languages and let Σ_1 be an alphabet. A *nondeterministic K-substitution* (or *nK-substitution*) τ is a mapping from Σ_1 into the set of K-languages which is extended to words in Σ_1^* by $\tau(\lambda) = \{\lambda\}$ and $\tau(a_1 \ldots a_n) = \tau(a_1) \ldots \tau(a_n)$, where $a_i \in \Sigma_1$ $(1 \le i \le n)$, or, equivalently,

$$\tau(a_1 \ldots a_n) = \{w_1 \ldots w_n \mid w_i \in \tau(a_i),\ 1 \le i \le n\}.$$

The mapping τ is extended to languages L_0 over Σ_1 by

$$\tau(L_0) = \cup\{\tau(w) \mid w \in L_0\}.$$ □

Notice that in case the family K equals *ONE*, *FIN* or *REG*, an nK-substitution is known as a homomorphism, finite substitution and regular substitution, respectively.

The addition of the adjective "nondeterministic" suggests that we can also consider deterministic substitutions [AsvEng77, EngSch]. The difference with the usual (nondeterministic) substitution – the additional "nondeterministic" may be omitted – is that in a deterministic K-substitution τ we choose in advance for each letter a in Σ_1 a fixed word w_a from the language $\tau(a)$ ($\tau(a)$ is a language in the family K). Then in the application of τ to a word ω each occurrence of a is replaced by w_a. The choice of the words w_a determines a homomorphism $h : \Sigma_1 \to \Sigma_2^*$. Therefore $\tau(\omega)$ is defined to be equal to the set of the images of all homomorphisms $h : \Sigma_1 \to \Sigma_2^*$ such that $h(a)$ is in $\tau(a)$. We define this formally.

Definition 3.6. Let K be a family of languages and let Σ_1 be an alphabet. A *deterministic K-substitution* (or *dK-substitution*) τ is a mapping from Σ_1 into the set of K-languages. It is extended to words in Σ_1^* by $\tau(\lambda) = \{\lambda\}$ and

$$\tau(a_1 \ldots a_n) = \{h(a_1) \ldots h(a_n) \mid h \text{ is a homomorphism such that } h(a) \in \tau(a) \text{ for each } a \in \Sigma_1\},\ \text{where } a_i \in \Sigma_1\ \ (1 \le i \le n).$$

The extension of τ to languages L_0 over Σ_1 is defined by

$$\tau(L_0) = \cup\{\tau(w) \mid w \in L_0\}.$$ □

From this definition it follows that $\tau(\omega) = \varnothing$ for each word ω in case $\tau(a) = \varnothing$ and at least one symbol a occurs in ω. It is also important to note that a dK-substitutions is not a special case of a nondeterministic

substitution, but they are both different generalizations of the notion of homomorphism. In fact, a homomorphism is both a d*ONE*-substitution and an n*ONE*-substitution.

Definition 3.7. A family F is closed under nK-substitution [dK-substitution] if for each language L_0 in F and each nK-substitution [dK-substitution, respectively] τ the language $\tau(L_0)$ is in F. In case the family K equals the family F, the we say that F is closed under (n-)substitution [d-substitution, respectively]. □

The following proposition shows that under weak assumptions on K the family of languages $RBLB_{r,f,\mathrm{OI}}(K)$ is closed under n-substitution.

Proposition 3.8. *Let K be a family closed under isomorphism such that $SYMBOL \cup \{\emptyset\} \subseteq K$. Then $RBLB_{r,f,\mathrm{OI}}(K)$ is closed under nondeterministic substitution.*

Proof. Let $L_1 = L_{r,\mathrm{OI}}(G_1, C_1)$ be a language in $RBLB_{r,f,\mathrm{OI}}(K)$, where $G_1 = (\Phi_{(1)}, \Psi_{(1)}, \Sigma_1, X_1, P_{(1)}, S_1)$. Let $\Sigma_1 = \{a_1, \ldots, a_N\}$, and let $\sigma : \Sigma_1 \to 2^{\Sigma^*}$ be a nondeterministic $RBLB_{r,f,\mathrm{OI}}(K)$-substitution, such that for each a in Σ_1 the language $\sigma(a)$ is generated by the (r,f,OI,REG,K)-belb grammar (G_a, C_a), where $G_a = (\Phi_{(a)}, \Psi_{(a)}, \Sigma_a, X_a, P_{(a)}, S_a)$. We construct an (r,f,OI,REG,K)-belb grammar (G,C) with underlying grammar $G = (\Phi, \Psi, \Sigma, X, P, S)$ such that $\sigma(L_1) = L_{r,\mathrm{OI}}(G,C)$.

Essentially, we use the terminals in Σ_1 of (G_1, C_1) as variables in (G,C) via a transformation which associates with each a in Σ_1 a corresponding variable y_a in X. Each terminal a in Σ_1 which occurs in the K-languages at the right-hand side of the productions of type I.3.3.1(iii) in (G_1, C_1) is replaced by the variable y_a. The original start symbol S_1 in (G_1, C_1) is transformed into S_1' such that S_1' has rank N. The new start symbol S is used in the new initial production $S \to S_1'(\psi_{a_1}, \ldots, \psi_{a_N})$. The other productions of (G,C) are obtained from those in (G_1, C_1) by adorning them with additional variables $y_{a_1}, \ldots, y_{a_N}$. Then throughout each derivation the language names $\psi_{a_1}, \ldots, \psi_{a_N}$ will be passed on downwards. By applying reductions $\psi_a \to S_a$ followed by a control word from C_a (where $a \in \Sigma_1$) in the proper way, each $a \in \Sigma_1$ in a word from $L_{r,\mathrm{OI}}(G_1, C_1)$ is substituted by the $RBLB_{r,f,\mathrm{OI}}(K)$-language $\sigma(a)$. Formally, we perform the construction of (G,C) in the following way.

Assume that the sets $\Phi_{(a)}$ with $a \in \Sigma_1$ are mutually disjoint. Let this property hold for the sets $\Psi_{(a)}$ and for the sets X_a too; in both cases a varies over Σ_1. Then the alphabets Φ and Ψ are defined by

$$\Phi_0 = \{S\} \cup \bigcup \{\Phi_{(a)0} \mid a \in \Sigma_1\},$$

$\Phi_n = \cup\{\Phi_{(a)n} \mid a \in \Sigma_1\}$ for each n with $1 \le n < N$,

$\Phi_{n+N} = \{A' \mid A \in \Phi_{(1)n}\} \cup \cup\{\Phi_{(a)n+N} \mid a \in \Sigma_1\}$ for each n $(n \ge 0)$,

and

$\Psi_0 = \{\psi_a \mid a \in \Sigma_1\} \cup \cup\{\Psi_{(a)0} \mid a \in \Sigma_1\}$,

$\Psi_n = \cup\{\Psi_{(a)n} \mid a \in \Sigma_1\}$ for each n with $1 \le n < N$,

$\Psi_{n+N} = \{\psi' \mid \psi \in \Psi_{(1)n}\} \cup \{\phi_{n,a} \mid a \in \Sigma_1\} \cup \cup\{\Psi_{(a)n+N} \mid a \in \Sigma_1\}$ for each n $(n \ge 0)$.

Define $X = X_1 \cup \{y_a \mid a \in \Sigma_1\} \cup \cup\{X_a \mid a \in \Sigma_1\}$. Let the sets U_1, U_2, and U_3 be defined in the following way, where $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{y} = (y_{a_1}, \ldots, y_{a_N})$.

$U_1 =$

$$\{A'(\vec{x},\vec{y}) \to B'(\psi_1'(\vec{x},\vec{y}), \ldots, \psi_k'(\vec{x},\vec{y}), \phi_{n,a_1}(\vec{x},\vec{y}), \ldots, \phi_{n,a_N}(\vec{x},\vec{y})) \mid$$

$$A' \in \Phi_{n+N},\ B' \in \Phi_{k+N},\ \phi_{n,a} \in \Psi_{n+N},\ a \in \Sigma_1,\ \psi_i' \in \Psi_{n+N},\ 1 \le i \le k,\ n \ge 0\},$$

$$U_2 = \{A'(\vec{x}, \vec{y}) \to \psi'(\vec{x}, \vec{y}) \mid A' \in \Phi_{n+N}, \psi' \in \Psi_{n+N},\ n \ge 0\},$$

$$U_3 = \{\psi'(\vec{x}, \vec{y}) \to i(L_0) \mid \psi' \in \Psi_{n+N}, \psi(\vec{x}) \to L_0 \in P_{(1)3},\ n \ge 0\},$$

where $i : \Sigma_1 \cup X_1 \to X$ is the isomorphism defined by

$$i(x) = x \qquad \text{if } x \in X_1,$$
$$i(a) = y_a \qquad \text{if } a \in \Sigma_1.$$

Let $U = U_1 \cup U_2 \cup U_3 \cup P_{\Sigma_1} \cup P_\phi$, where $P_{\Sigma_1} = \{S_a \to \psi_a \mid a \in \Sigma_1\}$ and $P_\phi = \{\phi_{n,a}(\vec{x}, \vec{y}) \to \{y_a\} \mid a \in \Sigma_1, n \ge 0\}$. Define the regular substitution $g : P_{(1)} \to 2^{U^*}$ by

$$g(A(\vec{x}) \to B(\psi_1(\vec{x}), \ldots, \psi_k(\vec{x}))) =$$

$$\{A'(\vec{x},\vec{y}) \to B'(\psi_1'(\vec{x},\vec{y}), \ldots, \psi_k'(\vec{x},\vec{y}), \phi_{n,a_1}(\vec{x},\vec{y}), \ldots, \phi_{n,a_N}(\vec{x},\vec{y}))\},$$

$$g(A(\vec{x}) \to \psi(\vec{x})) = \{A'(\vec{x}, \vec{y}) \to \psi'(\vec{x}, \vec{y})\},$$

$$g(\psi(\vec{x}) \to L_0) = \{(\psi'(\vec{x}, \vec{y}) \to i(L_0))(P_\phi^*(\overline{P_{\Sigma_1}} \cup \{C_a \mid a \in \Sigma_1\})^*)^*\}.$$

We define the set $P_\varnothing$ to be equal to $\{\psi_a \to \varnothing \mid a \in \Sigma_1\}$ in order to satisfy the condition that G ought to be an (OI,K)-elb grammar. Furthermore, if π is in $P_{(1)1} \cup P_{(1)2}$, then $g(\bar{\pi})$ is defined as $g(\bar{\pi}) = \overline{g(\pi)}$.

Let $\pi_0 = S \to S_1'(\psi_{a_1}, \ldots, \psi_{a_N})$. Then P is defined by

$$P = \cup\{P_{(a)} \mid a \in \Sigma_1\} \cup \{\psi'(\vec{x}, \vec{y}) \to i(L_0) \mid (\psi(\vec{x}) \to L_0) \in P_{(1)3}\} \cup$$
$$P_{\Sigma_1} \cup P_\phi \cup \{\pi_0\} \cup g(P_{(1)1} \cup P_{(1)2}) \cup P_\varnothing.$$

Finally, as the control language we take C equal to $\pi_0 g(C_1)$. □

We recall the following concepts. A family of languages is called *nontrivial* if it contains a language which differs from $\emptyset$ and from $\{\lambda\}$. A *full Abstract Family of Languages* or *full AFL* is a nontrivial family of languages which is closed under union, concatenation, Kleene +, homomorphism, inverse homomorphism and intersection with regular languages.

Corollary 3.9. *Let K be a nontrivial family closed under ngsm mappings. Then* $RBLB_{r,f,\mathrm{OI}}(K)$ *is a full substitution-closed AFL.*

Proof. Recall that it is sufficient to prove closure under intersection with regular languages, regular substitution and union with a regular set in order to prove closure under inverse homomorphism [Gin]. We can easily show by the inclusion $LB_{\mathrm{OI}}(K) \subseteq RBLB_{r,f,\mathrm{OI}}(K)$ that under the premisses on K the regular languages are included in $RBLB_{r,f,\mathrm{OI}}(K)$. Then the statement follows immediately from Propositions 3.1, 3.3, and 3.8, and Corollary 3.4. □

For the IO-mode closure under K-substitution or even under finite substitution is unlikely. On the other hand we can establish closure under intersection with regular languages and under deterministic substitution.

Proposition 3.10. *Let K be a family closed under intersection with regular languages. Then* $RBLB_{r,f,\mathrm{IO}}(K)$ *is closed under intersection with regular languages.*

Proof. The proof is based on a modification of the technique of factored grammars [Fis68a]. Recall that each regular set R equals the (finite) union of a number of congruence classes which corresponds to a congruence relation $\equiv$ – with respect to concatenation – over Σ^* of finite index; cf. [Sal73]. Starting from an (r,f,IO,REG,K)-belb grammar (G_1,C_1) we will construct an (r,f,IO,REG,K)-belb grammar (G,C) such that in (G,C) we can tell just by looking at a nonterminal or a language name to which congruence class its arguments must belong if this nonterminal or language name ever appears in a sentential form. We also can determine to which congruence class any string generated by this nonterminal will belong. Therefore, we transform the grammar (G_1,C_1) in the following way. Each nonterminal and language name is adorned with $n+1$ congruence classes $u_0,\ldots,u_n$, where n is the rank of that nonterminal or language name. However, an exception is made for the start symbol S_1, which is left unchanged. The congruence class of the resulting terminal string generated by the transformed nonterminal – if any – is equal to u_0.

The transformation of productions is arranged as follows. In connection with the exception made for the start symbol, the congruence class u_0 of the nonterminal on the right-hand side of an initial production ought to be taken from the finite number of congruence classes, the union of which

equals R. The transformation of the remaining productions is such that with respect to the nonterminal or language name to the left-hand side we can freely choose the congruence classes $u_0, \ldots, u_n$. Then the congruence class u_0 of the nonterminal or language name on top level of the right-hand side is equal to the corresponding congruence class on the left-hand side. And in case of non-initial productions of type I.3.3.1(i) and productions of type I.3.3.1(ii) the congruence class of each argument of the nonterminal on the left-hand side determines the congruence class of the corresponding argument of the language name on the right-hand side.

Concerning the productions of type I.3.3.1(iii) we replace the language L by $L \cap R_{u_0, \vec{u}}$, where $\vec{u} = (u_1, \ldots, u_n)$. The regular set $R_{u_0, \vec{u}}$ consists of those words in $(\Sigma \cup \{x_1, \ldots, x_n\})^*$ such that substituting an element from the congruence class u_i for x_i $(1 \leq i \leq n)$ results in a word from the congruence class u_0. After applying this transformation, a combination $u_0, \ldots, u_n$ of congruence classes combined with a production $\psi(x_1, \ldots, x_n) \to L_0$ of type I.3.3.1(iii) gives a blocked derivation in case the intersection of the language L and the regular set $R_{u_0, \vec{u}}$ is empty. Viz., in that case there are no rules to rewrite the language name $[\psi, u_0, \vec{u}]$, and this particular guess of the grammar gives no contribution to the language generated by (G, C).

It is left to the reader to check that the construction below formalizes the ideas presented above, and that the resulting grammar (G, C) generates the language $L_{r,\mathrm{IO}}(G_1, C_1) \cap R$.

Let $G_1 = (\Phi_{(1)}, \Psi_{(1)}, \Sigma, X_1, P_{(1)}, S_1)$ and $C_1 \subseteq (P_{(1)} \cup \bar{P}_{(1)})^*$. Let R be a regular language accepted by the deterministic finite automaton $M = (Q, \Sigma, \delta, q_0, F)$. The relation $\equiv$ on $\Sigma^* \times \Sigma^*$ is defined by

$$x \equiv y \quad \text{if and only if} \quad \forall q \in Q \,.\, \delta(q, x) = \delta(q, y).$$

Because R is regular, it is possible to partition Σ^* by $\equiv$ into a finite number of congruence classes. Let $\Sigma^*/\equiv$ denote the set of congruence classes $[t_1], \ldots, [t_k]$ induced by $\equiv$, where $t_1, \ldots, t_k$ are freely chosen but fixed representatives. Let $R_{\equiv}$ be equal to $\{[t_i] \mid \delta(q_0, t_i) \in F,\ 1 \leq i \leq k\}$. Then we have the identity $R = \cup R_{\equiv}$. Define the sets $R_{u_0, \vec{u}}$, where $u_i \in \Sigma^*/\equiv$ $(0 \leq i \leq n)$ by

$$R_{u_0, \vec{u}} = h_{\vec{u}}^{-1}(u_0),$$

where $h_{\vec{u}} : (\Sigma \cup X_1)^* \to \Sigma^*$ is the homomorphism defined by

$h_{\vec{u}}(a) = a$ for each a in Σ,
$h_{\vec{u}}(x_i) = t_i$ for each x_i in X_1, where t_i equals $u_i = [t_i]$.

Note that $R_{u_0,\,\vec{u}}$ is regular.

Consider the following (r,f,IO,REG,K)-belb grammar (G,C) which generates the language $L_{r,\mathrm{IO}}(G_1,C_1)\cap R$. The underlying grammar G equals $(\Phi,\Psi,\Sigma,X_1,P,S_1)$, where P is given by $P=g(P_{(1)})$. The finite substitution g is defined in the following way for rules in $P_{(1)}$ of the types I.3.3.1(i), I.3.3.1(ii) and I.3.3.1(iii). Let $\vec{u}=(u_1,\ldots,u_n)$ and $\vec{v}=(v_1,\ldots,v_k)$. Then

$$g(S_1\to A(\psi_1,\ldots,\psi_n))=\{S_1\to[A,u_0,\vec{u}\,]([\psi_1,u_1],\ldots,[\psi_n,u_n])\mid u_0\in R_{\equiv},\ u_1,\ldots,u_n\in\Sigma^*/{\equiv}\},$$

$$g(A(\vec{x})\to B(\psi_1(\vec{x}),\ldots,\psi_k(\vec{x})))=$$

$$\{[A,u_0,\vec{u}\,](\vec{x})\to[B,u_0,\vec{v}\,]([\psi_1,v_1,\vec{u}\,](\vec{x}),\ldots,[\psi_k,v_k,\vec{u}\,](\vec{x}))\mid u_0,\ldots,u_n,v_1,\ldots,v_k\in\Sigma^*/{\equiv}\},$$

$$g(A(\vec{x})\to\psi(\vec{x}))=\{[A,u_0,\vec{u}\,](\vec{x})\to[\psi,u_0,\vec{u}\,](\vec{x})\mid u_0,\ldots,u_n\in\Sigma^*/{\equiv}\},$$

$$g(\psi(\vec{x})\to L_0)=\{[\psi,u_0,\vec{u}\,](\vec{x})\to L_0\cap R_{u_0,\,\vec{u}}\mid u_0,\ldots,u_n\in\Sigma^*/{\equiv}\}.$$

Since there are no terminal reductions involved, a reduction $\bar{\pi}$ is always a reduction associated with an argument-preserving production of type I.3.3.1(i) or I.3.3.1(ii). Therefore, we can define $g(\bar{\pi})$ to be equal to $\overline{g(\pi)}$.

The ranked alphabet Φ of nonterminals is given by

$$\Phi_n=\{[A,u_0,\vec{u}\,]\mid A\in\Phi_{(1)n},\ u_0,\ldots,u_n\in\Sigma^*/{\equiv}\}\ \text{ for each } n\ (n\geq 1),$$

$$\Phi_0=\{S_1\}\cup\{[A,u_0]\mid A\in\Phi_{(1)0}-\{S_1\},\ u_0\in\Sigma^*/{\equiv}\}.$$

The sets Ψ_n of language names are given for each n $(n\geq 0)$ by

$$\Psi_n=\{[\psi,u_0,\vec{u}\,]\mid\psi\in\Psi_{(1)n},\ u_0,\ldots,u_n\in\Sigma^*/{\equiv}\}.$$

Finally, as the control language we take $C=g(C_1)$. □

Although – as remarked before – for the IO-mode closure under finite substitution is unlikely, we have closure under deterministic substitution.

Proposition 3.11. *Let K be a family closed under isomorphism such that SYMBOL $\cup\{\varnothing\}\subseteq K$. Then $RBLB_{r,f,\mathrm{IO}}(K)$ is closed under deterministic substitution.*

Proof. Let L_1 be an $RBLB_{r,f,\mathrm{IO}}(K)$-language, i.e., $L_1=L_{r,\mathrm{IO}}(G_1,C_1)$, where $G_1=(\Phi_{(1)},\Psi_{(1)},\Sigma_1,X_1,P_{(1)},S_1)$. Let $\Sigma_1=\{a_1,\ldots,a_N\}$, and let $\sigma:\Sigma_1\to 2^{\Sigma^*}$ be a $RBLB_{r,f,\mathrm{IO}}(K)$-substitution, such that for each a in Σ_1 the

language $\sigma(a)$ is generated by the (r,f,IO,REG,K)-belb grammar (G_a,C_a), where $G_a = (\Phi_{(a)},\Psi_{(a)},\Sigma_a,X_a,P_{(a)},S_a)$. We construct an (r,f,IO,REG,K)-belb grammar (G,C) with underlying grammar $G = (\Phi,\Psi,\Sigma,X,P,S)$ such that $\sigma(L_1) = L_{r,\mathrm{IO}}(G,C)$.

The construction resembles much to the proof of Proposition 3.8. We use the terminals in Σ_1 of (G_1,C_1) as variables in (G,C). This is obtained via the isomorphism v which associates with each a in Σ_1 a corresponding variable y_a in X. Each terminal a in Σ_1 which occurs in the K-languages at the right-hand side of the productions of type I.3.3.1(iii) in (G_1,C_1) is replaced by the variable y_a. The original start symbol S_1 in (G_1,C_1) is transformed into S_1' such that S_1' has rank N. The new start symbol S of G is used in the new initial production π_0 equal to $S \to S_1'(\psi_{a_1},\ldots,\psi_{a_N})$. The other productions of (G,C) are obtained from those in (G_1,C_1) by adorning them with additional variables $y_{a_1},\ldots,y_{a_N}$. A derivation in (G,C) starts with the initial production, followed by a sequence of control strings from the set $\cup\{(\psi_a \to S_a)\,C_a \mid a \in \Sigma_1\}$, until we have obtained a term in $Term(G)$ of the form $S_1'(w_1,\ldots,w_N)$, where $w_i \in \sigma(a_i)$ $(1 \le i \le N)$. From then on we can follow a derivation according to C_1, where the construction is such that each letter a_i in Σ_1 $(1 \le i \le N)$ is replaced by a fixed word w_i from $\sigma(a_i)$. The formal construction is as follows.

We assume that the sets $\Phi_{(a)}$ with $a \in \Sigma_1$ are mutually disjoint. Let the sets $\Psi_{(a)}$ and the sets X_a, where a varies over Σ_1, possess this property too. Then the alphabets Φ and Ψ are defined by

$$\Phi_0 = \{S\} \cup \cup\{\Phi_{(a)0} \mid a \in \Sigma_1\},$$

$$\Phi_n = \cup\{\Phi_{(a)n} \mid a \in \Sigma_1\} \text{ for each } n \text{ with } 1 \le n < N,$$

$$\Phi_{n+N} = \{A' \mid A \in \Phi_{(1)n}\} \cup \cup\{\Phi_{(a)n+N} \mid a \in \Sigma_1\} \text{ for each } n\ (n \ge 0),$$

and

$$\Psi_0 = \{\psi_a \mid a \in \Sigma_1\} \cup \cup\{\Psi_{(a)0} \mid a \in \Sigma_1\},$$

$$\Psi_n = \cup\{\Psi_{(a)n} \mid a \in \Sigma_1\} \text{ for each } n \text{ with } 1 \le n < N,$$

$$\Psi_{n+N} = \{\psi' \mid \psi \in \Psi_{(1)n}\} \cup \{\phi_{n,a} \mid a \in \Sigma_1\} \cup \cup\{\Psi_{(a)n+N} \mid a \in \Sigma_1\} \text{ for each } n\ (n \ge 0).$$

Define $X = X_1 \cup \{y_a \mid a \in \Sigma_1\} \cup \cup\{X_a \mid a \in \Sigma_1\}$. Let the sets U_1, U_2, and U_3 be defined in the following way, where $\vec{x} = (x_1,\ldots,x_n)$ and $\vec{y} = (y_{a_1},\ldots,y_{a_N})$.

$U_1 =$

$$\{A'(\vec{x},\vec{y}) \to B'(\psi_1'(\vec{x},\vec{y}),\ldots,\psi_k'(\vec{x},\vec{y}),\phi_{n,a_1}(\vec{x},\vec{y}),\ldots,\phi_{n,a_N}(\vec{x},\vec{y})) \mid$$

$$A' \in \Phi_{n+N},\ B' \in \Phi_{k+N},\ \phi_{n,a} \in \Psi_{n+N},\ a \in \Sigma_1,\ \psi_i' \in \Psi_{n+N},\ 1 \le i \le k,\ n \ge 0\},$$

$$U_2 = \{A'(\vec{x}, \vec{y}) \to \psi'(\vec{x}, \vec{y}) \mid A' \in \Phi_{n+N}, \psi' \in \Psi_{n+N}, n \geq 0\},$$

$$U_3 = \{\psi'(\vec{x}, \vec{y}) \to v(L_0) \mid \psi' \in \Psi_{n+N}, \psi(\vec{x}) \to L_0 \in P_{(1)3}, n \geq 0\},$$

where $v : \Sigma_1 \cup X_1 \to X$ is the isomorphism defined by

$$\begin{aligned} v(x) &= x \qquad \text{if } x \in X_1, \\ v(a) &= y_a \qquad \text{if } a \in \Sigma_1. \end{aligned}$$

Let $U = U_1 \cup U_2 \cup U_3 \cup P_{\Sigma_1} \cup P_\phi$, where $P_{\Sigma_1} = \{S_a \to \psi_a \mid a \in \Sigma_1\}$ and $P_\phi = \{\phi_{n,a}(\vec{x}, \vec{y}) \to \{y_a\} \mid a \in \Sigma_1, n \geq 0\}$. Define the regular substitution $g : P_{(1)} \to 2^{U^*}$ by

$$g(A(\vec{x}) \to B(\psi_1(\vec{x}), \ldots, \psi_k(\vec{x}))) =$$

$$\{A'(\vec{x}, \vec{y}) \to (B'(\psi_1'(\vec{x}, \vec{y}), \ldots, \psi_k'(\vec{x}, \vec{y}), \phi_{n,a_1}(\vec{x}, \vec{y}), \ldots,$$

$$\phi_{n,a_N}(\vec{x}, \vec{y})))P_\phi^*\},$$

$$g(A(\vec{x}) \to \psi(\vec{x})) = \{A'(\vec{x}, \vec{y}) \to \psi'(\vec{x}, \vec{y})\},$$

$$g(\psi(\vec{x}) \to L_0) = \{\psi'(\vec{x}, \vec{y}) \to v(L_0)\}.$$

In order to satisfy the condition that G ought to be an (IO,K)-elb grammar, we define the set $P_\varnothing$ to be equal to $\{\psi_a \to \varnothing \mid a \in \Sigma_1\}$. Furthermore, if π is in $P_{(1)1} \cup P_{(1)2}$, then $g(\bar{\pi})$ is defined as $g(\bar{\pi}) = \overline{g(\pi)}$.

Let $\pi_0 = S \to S_1'(\psi_{a_1}, \ldots, \psi_{a_N})$. Then P is defined by

$$P = \cup\{P_{(a)} \mid a \in \Sigma_1\} \cup \{\psi'(\vec{x}, \vec{y}) \to v(L_0) \mid (\psi(\vec{x}) \to L_0) \in P_{(1)3}\} \cup$$

$$P_{\Sigma_1} \cup P_\phi \cup \{\pi_0\} \cup g(P_{(1)1} \cup P_{(1)2}) \cup P_\varnothing.$$

Finally, as the control language we take C equal to

$$\pi_0\{(\psi_a \to S_a)C_a \mid a \in \Sigma_1\}^* g(C_1). \qquad \square$$

Remark that it follows from Proposition 3.11 that the language family $RBLB_{r,f,\mathrm{IO}}(K)$ is closed under homomorphism in case K is closed under isomorphism.

Recall that a language family is a *full Quasi Abstract Family of Languages* or *full QAFL* [AsvEng77], if it is a family that contains at least one *SYMBOL*-language and that is closed under the regular operations (union, concatenation, and Kleene $*$), intersection with regular languages, and homomorphism.

Corollary 3.12. *Let K be a family with $K \supseteq SYMBOL$, and let K be closed under left or right-marking, intersection with regular languages, and isomorphism. Then the family $RBLB_{r,f,\mathrm{IO}}(K)$ is a full QAFL closed under*

deterministic substitution. □

4. Generating Power of (r,f,m,REG,K)-belb Grammars

In this section we determine a lower bound on the language generating capacity of (r,f,m,REG,K)-belb grammars. First, we establish some results which are analogously to the non-bidirectional (unidirectional) case. Let the family $BLB_{r,f,m}(K)$ denote the family of languages generated by (r,f,m,REG,K)-belb grammars (G,C) in which $C = (P \cup \bar{P})^*$. Such uncontrolled bidirectional grammars are called (r,f,m,K)-*belb grammars* in the sequel. Consequently, the control language C will be omitted in the pair (G,C), so that we denote such a grammar by a single tuple G.

Lemma 4.1.

(i) $RBLB_{r,f,\mathrm{IO}}(\varnothing NE) = RBLB_{r,f,\mathrm{IO}}(FIN)$.

(ii) $BLB_{r,f,\mathrm{IO}}(\varnothing NE) = BLB_{r,f,\mathrm{IO}}(FIN)$.

(iii) For each family K we have $BLB_{r,f,m}(K) \subseteq RBLB_{r,f,m}(K)$, where either $m = \mathrm{OI}$ or $m = \mathrm{IO}$.

Proof. (i). The inclusion from left to right is obvious. The converse inclusion can be shown by replacing each production $\psi(x_1,\ldots,x_n) \to L_0$ in an $(r,f,\mathrm{IO},REG,FIN)$-belb grammar by productions $\psi_i(x_1,\ldots,x_n) \to \{\eta_i\}$, where it is understood that $L_0 = \{\eta_1,\ldots,\eta_k\}$ for some $k \geq 0$. Here $k = 0$ means that $L_0 = \varnothing$, in which case no replacement ought to be made. As a consequence, each rule ρ in which $\psi(\vec{x})$ occurs l-times has to be replaced by k^l rules covering all combinations of ψ_i's $(1 \leq i \leq k)$ possible in ρ. The corresponding alterations in the control language C are allowed, for REG is closed under finite substitution.

(ii) and (iii). Obvious. □

In the following proposition we show that the family IO is included in $BLB_{r,f,\mathrm{IO}}(\varnothing NE)$. With respect to an m-macro grammar G equal to (Φ,Σ,X,P,S) we define for each $A \in \Phi$ the finite (possibly empty) language $L_{A,G}$ over $\Sigma \cup X$ by $L_{A,G} = \{\eta \in (\Sigma \cup X)^* \mid \exists \pi \in P \,.\, \pi = A(\vec{x}) \to \eta\}$. Note that $L_{A,G}$ does not depend on the mode m.

Proposition 4.2. *The family* IO *of IO-macro languages is included in the families* $BLB_{r,f,\mathrm{IO}}(\varnothing NE)$ and $RBLB_{r,f,\mathrm{IO}}(\varnothing NE)$.

Proof. Let L_0 be an IO-macro language generated by the grammar $G_1 = (\Phi_{(1)},\Sigma,X,P_{(1)},S)$. We assume G_1 is in IO standard form [Fis68a]; i.e., each production is argument-preserving and it has either the form

(i) $A(x_1,\ldots,x_n) \to B(D(y_1,\ldots,y_l),z_2,\ldots,z_k)$, where $A \in \Phi_{(1)n}$, $B \in \Phi_{(1)k}$ $(k \geq 1)$, $D \in \Phi_{(1)l}$ and $y_1,\ldots,y_l,z_2,\ldots,z_k \in X$, or

(*ii*) $A(x_1,\ldots,x_n) \to \eta$, where $\eta \in (\Sigma \cup X)^*$.

We construct an (r, f, IO, *FIN*)-belb grammar G with underlying grammar $G = (\Phi, \Psi, \Sigma, X, P, S)$ such that $L_{r,\mathrm{IO}}(G) = L_{\mathrm{IO}}(G_1)$ as follows. Starting from $P = \varnothing$, for each production π in $P_{(1)}$ we add to P a sequence of productions. If π is of the form (*i*), then we add productions $p_{ABD\pi}$, $p_{D'\pi D}$ and $p_{D'\pi}$ to P, where $D' \in \Phi_n$ ($n \geq 0$) and

$$p_{ABD\pi} = A(x_1,\ldots,x_n) \to B(\psi_{D\pi}(\vec{x}), \psi_{z_2}(\vec{x}), \ldots, \psi_{z_k}(\vec{x})),$$

$$p_{D'\pi D} = D'_\pi(x_1,\ldots,x_n) \to D(\psi_{y_1}(\vec{x}), \ldots, \psi_{y_l}(\vec{x})),$$

$$p_{D'\pi} = D'_\pi(x_1,\ldots,x_n) \to \psi_{D\pi}(x_1,\ldots,x_n).$$

If π is of the form (*ii*), then we add to P the productions π_A and $\pi_{A''}$, where

$$\pi_A = A(x_1,\ldots,x_n) \to \psi_A(x_1,\ldots,x_n),$$

$$\pi_{A''} = \psi_A(x_1,\ldots,x_n) \to L_{A,G_1}.$$

Furthermore, we add to P the elements of the set P_X, consisting of all productions $\psi_x(x_1,\ldots,x_n) \to \{x\}$, with $x \in X$ and x occurs in $\vec{x}$. A nonterminal D ought to be expanded by the corresponding language name ψ_D. Therefore, we add to P all productions $\psi_{D\pi}(\vec{x}) \to \varnothing$, in case D occurs in the argument list of the right-hand side of a nested production π in G_1. From the construction of P one can easily determine Φ and Ψ. We observe that a production of the form (*i*) is simulated in G by some element of $\{p_{ABD\pi}\}P_X^*\{\bar{p}_{D'\pi}\, p_{D'\pi D}\}P_X^*$. In addition, a production of the form (*ii*) is simulated in G by $\pi_A\pi_{A''}$. However, it is not necessary to provide G with a control language in order to generate the language L_0. The correct order of application is arranged implicitly by the derivation mode. Therefore, we can take for C the trivial control language $(P \cup \bar{P})^*$.

So far we have shown that $IO \subseteq BLB_{r,f,\mathrm{IO}}(FIN)$. The conclusion now follows from Lemma 4.1. □

Corollary 4.3. *If K is a language family that includes ∅NE, then the families* $BLB_{r,f,\mathrm{IO}}(K)$ *and* $RBLB_{r,f,\mathrm{IO}}(K)$ *both contain all IO-macro languages.* □

For *m* equal to OI an analogous result holds.

Proposition 4.4. *The family* OI *of OI-macro languages is included in the families* $BLB_{r,f,\mathrm{OI}}(\varnothing NE)$ and $RBLB_{r,f,\mathrm{OI}}(\varnothing NE)$.

Proof. Let L_0 be an OI-macro language generated by the grammar $G_1 = (\Phi_{(1)}, \Sigma, X, P_{(1)}, S)$. Assume G_1 is in OI standard form [Fis68a]; that means that each production has either the form

(*i*) $A(x_1,\ldots,x_n) \to B(D_1(x_1,\ldots,x_n),\ldots,D_k(x_1,\ldots,x_n))$, with $k,n \geq 0$,

or

(*ii*) $A(x_1,\ldots,x_n) \to \eta$, where $\eta \in (\Sigma \cup X)^*$ and $n \geq 0$.

Furthermore, we assume that in G_1 the symbol S only occurs at the left-hand side of productions of the form (*i*). This is no loss of generality, since we can eventually transform the grammar G_1 into the equivalent grammar G_2 equal to $(\Phi_{(2)},\Sigma,X_{(2)},P_{(2)},S')$, where

$$\Phi_{(2)} = \Phi_{(1)} \cup \{S', S''\}, \qquad X' = X \cup \{x\}, \text{ and}$$
$$P_{(2)} = P_{(1)} \cup \{S' \to S''(S),\ S''(x) \to x\}.$$

Analogously to Proposition 4.2 we construct an $(r,f,\mathrm{OI},\varnothing NE)$-belb grammar G with $G = (\Phi,\Psi,\Sigma,X,P,S)$ such that $L_{r,\mathrm{OI}}(G) = L_{\mathrm{OI}}(G_1)$ as follows. For each nested production π in $P_{(1)}$ of the form (*i*), we add a corresponding production π' to P, where π' is defined by

$$A(\vec{x}) \to B(\psi_{D_1}(\vec{x}),\ldots,\psi_{D_k}(\vec{x})).$$

Define $P'_{(1)i}$ by $P'_{(1)i} = \{\pi' \mid \pi \in P_{(1)}, \pi \text{ is of the form } (i)\}$.

Let Θ be the set of all nonterminals in $\Phi_{(1)}$ that occur in the argument list at the right-hand side of a nested production of the form (*i*). For each D in Θ we introduce a language name ψ_D and a production of type I.3.3.1(ii), viz. $D(\vec{x}) \to \psi_D(\vec{x})$. In addition, we define for each such D a production of type I.3.3.1(iii) by $\psi_D(\vec{x}) \to \varnothing$. The two sets of all productions of the form I.3.3.1(ii) and I.3.3.1(iii) obtained in this way are denoted by P_Θ and $P_\varnothing$, respectively. Thus

$$P_\Theta = \{D(\vec{x}) \to \psi_D(\vec{x}) \mid D \in \Phi_{(1)}\}, \text{ and}$$
$$P_\varnothing = \{\psi_D(\vec{x}) \to \varnothing \mid D \in \Phi_{(1)}\}.$$

Next, we define P by

$$P = P'_{(1)i} \cup P_\Theta \cup P_\varnothing \cup \{A(\vec{x}) \to \psi_\eta(\vec{x}),\ \psi_\eta(\vec{x}) \to \{\eta\} \mid A(\vec{x}) \to \eta \in P_{(1)}\}$$

We take the set of nonterminals Φ equal to $\Phi_{(1)}$ and the set of language names Ψ is defined by

$$\Psi_n = \{\psi_\eta \mid A(\vec{x}) \to \eta \in P_{(1)}, A \neq S\} \cup \{\psi_D \mid D \in \Theta \cap \Phi_{(1)n}\},$$

for each n ($n \geq 0$).

The application of a nested production of the form (*i*) is simulated by the corresponding production $A(\vec{x}) \to B(\psi_{D_1}(\vec{x}),\ldots,\psi_{D_k}(\vec{x}))$ in G. In case a language name ψ_D percolates at top level, it has to be rewritten into

the corresponding nonterminal D. Thus a terminal production π equal to $B(\vec{x}) \to \eta$, with $\eta \in (\Sigma \cup X)^*$, in G_1 is simulated by the sequence of rules

$$(B(\vec{x}) \to \psi_\eta(\vec{x}))(\psi_\eta(\vec{x}) \to \{\eta\})(\bar{P}_\Theta \cup \{\lambda\}).$$

The additional $\{\lambda\}$ in this sequence is necessary to cover the case in which B has no arguments. Note that the trivial control language suffices, i.e., we can take C equal to $(P \cup \bar{P})^*$. □

Corollary 4.5. *If K is a language family that includes $\varnothing$NE, then the families $BLB_{r,f,\mathrm{OI}}(K)$ and $RBLB_{r,f,\mathrm{OI}}(K)$ both contain all OI-macro languages.* □

In the remaining part of this section we show that the language family $RBLB_{r,f,\mathrm{OI}}(OI)$ equals the family OI; cf. Theorem 4.13.

Let G be an OI-macro grammar. Then we define the language $L_{r,\mathrm{OI}}(G)$ over Σ^* by

$$L_{r,\mathrm{OI}}(G) = \{w \in \Sigma^* \mid S \Rightarrow^*_{r,\mathrm{OI}} w\},$$

where $\alpha \Rightarrow_{r,\mathrm{OI}} \beta$ holds if and only if β is obtained from α by a single right-most (OI) derivation step. The strings α and β are terms over the alphabet of G. For IO and OI-macro grammars, the right-most derivation relation can be defined analogously to Definition 2.6. An OI-macro grammar provided with right-most rewriting will be called an (r,OI)-macro grammar, and the language $L_{r,\mathrm{OI}}(G)$ will be called an (r,OI)-macro language. Let OI_r denote the family of (r,OI)-macro languages. In addition, let $OI_r(REG)$ denote the family of languages generated by regularly controlled (r,OI)-macro grammars. Then we can prove the following result.

Proposition 4.6. *The family $RBLB_{r,f,\mathrm{OI}}(OI)$ is included in the family $OI_r(REG)$.*

Proof. Let L_0 be generated by the (r,f,OI,REG,OI)-belb grammar (G_1,C_1), where $G_1 = (\Phi_{(1)},\Psi,\Sigma,X_{(1)},P_{(1)},S)$. We construct a regularly controlled OI-macro grammar (G,C) with $G = (\Phi,\Sigma,X,P,S)$ such that $L_0 = L_{r,\mathrm{OI}}(G,C)$. Starting with $P = \varnothing$ we add for each rule in $P_{(1)} \cup \bar{P}_{(1)}$ one or more productions to P as follows. If $\rho \in P_{(1)2} \cup \bar{P}_{(1)2} \cup P_{(1)1}$, then ρ is added to P. If ρ is in $P_{(1)3}$, then ρ is of the form $\psi(x_1,\ldots,x_n) \to L_\psi$. Let for each language name ψ in Ψ the language L_ψ be generated by some OI-macro grammar $G_\psi = (\Phi_\psi,\Sigma \cup \{x_1,\ldots,x_n\},Y_\psi,P_\psi,S_\psi)$. We assume that the alphabets of this finite number of grammars G_ψ are mutually disjoint. Then we add each production in P'_ψ to P, where P'_ψ equals $\{A'(\vec{y},\vec{x}) \to t \mid A(\vec{y}) \to t \in P_\psi\}$, and we define $P'_\Psi = \cup\{P'_\psi \mid \psi \in \Psi\}$. Next, we add to P productions $\psi(\vec{x}) \to S_\psi(\vec{x})$, where $\psi \in \Psi$. Finally, if the rule ρ is in $\bar{P}_{(1)1}$, then ρ is a reduction of the form $B(\psi_1(\vec{x}),\ldots,\psi_k(\vec{x})) \to A(\vec{x})$. First consider the case that A is in $\Phi_{(1)0}$.

Then the production π_{BA} equal to $B(\vec{x}) \to A$ is added to P. Secondly, if A is in $\Phi_{(1)n}$ $(n \geq 1)$, then we add productions π_{B1} equal to $B(\vec{x}) \to x_1$ (Remember that $k \geq 1$.) and $\pi_{\psi_1 A}$ equal to $\psi_1(\vec{x}) \to A(\vec{x})$ to P.

Now we define P to be equal to

$$P_{(1)2} \cup \bar{P}_{(1)2} \cup P_{(1)1} \cup \cup \{\psi(\vec{x}) \to S_\psi(\vec{x}) \mid \psi(\vec{x}) \to L_\psi \in P_{(1)3}\} \cup P'_\Psi$$

$$\cup \{\pi_{\psi_1 A} \mid \exists B \in \Phi \cdot \exists \psi_2, \ldots, \exists \psi_k \in \Psi \cdot A(\vec{x}) \to B(\psi_1(\vec{x}), \ldots, \psi_k(\vec{x})) \in P_{(1)}\}$$

$$\cup \{\pi_{B1} \mid \exists A \in \Phi \cdot \exists \psi_1, \ldots, \exists \psi_k \in \Psi \cdot A(\vec{x}) \to B(\psi_1(\vec{x}), \ldots, \psi_k(\vec{x})) \in P_{(1)}\}$$

$$\cup \{\pi_{BA} \mid \exists \psi_1, \ldots, \exists \psi_k \in \Psi \cdot A(\vec{x}) \to B(\psi_1(\vec{x}), \ldots, \psi_k(\vec{x})) \in P_{(1)}\}.$$

Next define the regular substitution $\sigma : (P_{(1)} \cup (\overline{P_{(1)} - P_{(1)3}}))^* \to 2^{P^*}$ by

$$\sigma(A(\vec{x}) \to \psi(\vec{x})) = \{A(\vec{x}) \to \psi(\vec{x})\},$$

$$\sigma(\psi(\vec{x}) \to A(\vec{x})) = \{\psi(\vec{x}) \to A(\vec{x})\},$$

$$\sigma(\psi(\vec{x}) \to L_\psi) = \{\psi(\vec{x}) \to S_\psi(\vec{x})\} P'^*_\Psi,$$

$$\sigma(A(\vec{x}) \to B(\psi_1(\vec{x}), \ldots, \psi_k(\vec{x}))) = \{A(\vec{x}) \to B(\psi_1(\vec{x}), \ldots, \psi_k(\vec{x}))\},$$

$$\sigma(B(\psi_1(\vec{x}), \ldots, \psi_k(\vec{x})) \to A(\vec{x})) = \{\pi_{B1}\pi_{\psi_1 A}\},$$

$$\sigma(B(\psi_1, \ldots, \psi_k) \to A) = \{\pi_{BA}\}.$$

From P we obtain Φ and X in a straightforward way. It easily follows from the construction of G that $L_0 = L_{r,\mathrm{OI}}(G, \sigma(C_1))$. □

Let the number of occurrences of a symbol σ in a word w be denoted by $\#_\sigma(w)$.

Example 4.7. Consider the $(r, f, \mathrm{OI}, REG, \varnothing NE)$-belb grammar (G, C) of Example I.3.3.3, where $G = (\Phi, \Psi, \{0,1\}, X, P, S)$, $X = \{x\}$, $C = (P \cup \bar{P})^*$, and P consists of

$$\begin{array}{ll}
\pi_0 = S \to A(\psi_1), & \pi_6 = B \to \psi_1, \\
\pi_1 = A(x) \to A(\psi_2(x)), & \pi_7 = B \to D(\psi_1), \\
\pi_2 = A(x) \to \psi_3(x), & \pi_8 = D(x) \to \psi_4(x), \\
\pi_3 = \psi_3(x) \to \{x\}, & \pi_9 = \psi_4(x) \to \{0x\}, \\
\pi_4 = \psi_2(x) \to \{xx\}, & \pi_{10} = D(x) \to \psi_5(x), \\
\pi_5 = \psi_1 \to \{1\}, & \pi_{11} = \psi_5(x) \to \{x0\}.
\end{array}$$

According to Section 2, we have that $P_1 = \{\pi_0, \pi_1, \pi_7\}$, $P_2 = \{\pi_2, \pi_6, \pi_8\pi_{10}\}$, and $P_3 = P - (P_1 \cup P_2)$. Now we replace production π_5 by the production $\pi'_5 = \psi_1 \to L_{\psi_1}$, where

$$L_{\psi_1} = \{w \in \{a, 1\}^+ \mid \#_1(w) = 2^k,\ k \geq 0\}.$$

The resulting grammar (G',C') is an (r,f,OI,REG,OI)-belb grammar, where $G'=(\Phi,\Psi,\{0,1,a\},X,P',S)$, with $P'=P\cup\{\pi_5'\}-\{\pi_5\}$ and C' is the resulting variant of C. It is easy to see that the language generated by this grammar equals

$$L_1=\{w\in\{0,1,a\}^+\mid w=w_1\ldots w_k,\ k=2^l,\ l\geq 0,\ w_i \text{ is in } 0^*a^*(1a^*)^{m_i}0^*,\ m_i=2^{l_i},\ l_i\geq 0\ \}.$$

The language L_{ψ_1} can be generated by the OI-macro grammar $G_{\psi_1}=(\Phi_{\psi_1},\{1,a\},Y,P_{\psi_1},S_{\psi_1})$, where $\Phi_{\psi_1}=\{S_{\psi_1},F,H\}$, $Y=\{y\}$, and P_{ψ_1} consists of the productions

$$\begin{array}{lll} S_{\psi_1}\to F(H), & F(y)\to y, & H\to Ha,\\ F(y)\to F(yy), & H\to aH, & H\to 1. \end{array}$$

The languages $\{u_i\}$, with $\psi_i(x)\to\{u_i\}$ $(2\leq i\leq 5)$, can be generated by OI-macro grammars which have a single production $S_{\psi_i}\to u_i$ $(2\leq i\leq 5)$.

Using the construction given in the proof of Proposition 4.6, we obtain the following regularly controlled (r,OI)-macro grammar (G_1,C_1) that generates L_1, where $G_1=(\Phi_{(1)},\Sigma,X_{(1)},P_{(1)},S)$, and $P_{(1)}$ is formed by

$$P_1\cup P_2\cup\bar{P}_2\cup\{\psi_i(x)\to S_{\psi_i}(x)\mid 2\leq i\leq 5\}\cup\{S_{\psi_i}(x)\to u_i\mid 2\leq i\leq 5\}\cup P_{\psi_1}\cup$$
$$\cup\{\psi_1\to S_{\psi_1},A(x)\to S,\ D(x)\to B,\ A(x)\to x,\ \psi_2(x)\to A(x)\}.$$

We define the regular substitution $\sigma:(P_{(1)}\cup(\overline{P_{(1)}-P_{(1)3}}))^*\to 2^{P^*}$ by

$$\sigma(\rho)=\{\rho\}\quad\text{for each }\rho\in P_{(1)}\cup P_{(2)}\cup\bar{P}_{(2)},$$
$$\sigma(\psi_1\to L_{\psi_1})=\{\psi_1\to S_{\psi_1}\}P_{\psi_1}^*,$$
$$\sigma(\psi_i(x)\to L_{\psi_i})=\{(\psi_i(x)\to S_{\psi_i})(S_{\psi_i}\to u_i)\}\quad\text{for each } i\ (2\leq i\leq 5),$$
$$\sigma(A(\psi_1)\to S)=\{A(x)\to S\},$$
$$\sigma(D(\psi_1)\to B)=\{D(x)\to B\},$$
$$\sigma(A(\psi_2(x))\to A(x))=\{(A(x)\to x)(\psi_2(x)\to A(x))\}.$$

Finally, $\Phi_{(1)}=\Phi\cup\Psi\cup\Phi_{\psi_1}\cup\{S_{\psi_i}\mid 2\leq i\leq 5\}$, $X_{(1)}=X\cup Y$ and as the control language we define $C_1=\sigma(C')$. □

Next we give a characterization of the family $OI_r(REG)$. This is achieved by a proof method of Ginsburg and Spanier [GinSpa], who proved that the family of languages generated by regularly controlled context-free grammars provided with left-most derivation equals the family of context-free languages. First we give the obvious right-most version of Theorem 2.1

from [GinSpa] which is formulated in our notation.

Theorem 4.8. [GinSpa]. *Let K be a family of languages closed under λ-free regular substitution. Then the family of languages generated by K-controlled context-free grammars provided with right-most rewriting equals the family of all languages of the form $h(L_1 \cap L_2)$, where L_1 is in K, L_2 is a context-free language, and h is a homomorphism.* □

If we replace in Theorem 4.8 "context-free" by "OI-macro" everywhere, then an analogous statement still holds; cf. Theorem 4.11. The proof according to [GinSpa] of Theorem 4.8 uses closure under inverse homomorphism of the family of context-free languages. Although this closure property does hold for the family *OI* [Fis68a], it is sufficient to have closure under isomorphism. To prove Theorem 4.11 we need the following two lemmas.

Lemma 4.9. $OI_r = OI$.

Proof. The proof is analogously to the context-free case, which is well known; cf. for instance [LewPap]. □

For each OI-macro grammar $G = (\Phi, \Sigma, X, P, S)$ we define the OI-macro grammar $H(G)$ by $(\Phi, \Sigma_1, X, P_1, S)$, where $\Sigma_1 = \Sigma \cup P$ and

$$P_1 = \{A(\vec{x}) \to \eta\, \pi \mid \pi \in P,\ \pi = A(\vec{x}) \to \eta\}.$$

Lemma 4.10. *Let G be an OI-macro grammar (Φ, Σ, X, P, S) and C be a control language over P. Then there exists a λ-free regular substitution τ and a homomorphism h such that $L_{r,\mathrm{OI}}(G,C) = h(L_{r,\mathrm{OI}}(H(G)) \cap \tau(C^R))$.*

Proof. Similarly to the proof of Lemma 2.1 in [GinSpa]. Viz., the homomorphism $h : \Sigma_1^* \to \Sigma^*$ is defined by $h(\pi) = \lambda$ if $\pi \in P$, and $h(a) = a$ if $a \in \Sigma$. The λ-free regular substitution τ is defined by $\tau(\pi) = \Sigma^* \{\pi\} \Sigma^*$, for each $\pi \in P$. □

The following theorem has been adapted from Theorem 2.1 in [GinSpa].

Theorem 4.11. *Let K be a family of languages closed under reversal and λ-free regular substitution. Then the family of languages generated by K-controlled OI-macro grammars provided with right-most rewriting equals the family of all languages of the form $h(L_1 \cap L_2)$, where L_1 is in K, L_2 is an OI-macro language, and h is a homomorphism.*

Proof. By Lemma 4.10 there exists for each control language C in K and each OI-macro grammar G a λ-free regular substitution τ and a homomorphism h such that $L_{r,\mathrm{OI}}(G,C) = h(L_{r,\mathrm{OI}}(H(G)) \cap \tau(C^R))$. Then $\tau(C^R)$ is in K. The language $L_{r,\mathrm{OI}}(H(G))$ is an OI-macro language (Lemma 4.9), so $L_{r,\mathrm{OI}}(G,C)$ has the proper form.

Conversely, let L_1 and L_2 be languages over Σ_1, where L_1 is a K-language and L_2 an OI-macro language. Let $h : \Sigma_1^* \to \Sigma_2^*$ be a homomorphism.

We may assume $\Sigma_1 \cap \Sigma_2 = \emptyset$, which is shown as follows. Let $\bar{a}$ be a distinct symbol not in Σ_2 for each a in Σ_1, and let $\bar{\Sigma}_1 = \{\bar{a} \mid a \in \Sigma_1\}$. Let h_1 be the isomorphism from $\bar{\Sigma}_1$ into Σ_1 defined by $h_1(\bar{a}) = a$ for each $\bar{a}$ in Σ_1. Then h_1^{-1} is an isomorphism too. Let $\bar{L}_1 = h_1^{-1}(L_1)$ and $\bar{L}_2 = h_1^{-1}(L_2)$. Then $\bar{\Sigma}_1 \cap \Sigma_2 = \emptyset$, $\bar{L}_1$ is in K, $\bar{L}_2$ is an OI-macro language [Fis68a], hh_1 is a homomorphism and $h(L_1 \cap L_2) = hh_1(\bar{L}_1 \cap \bar{L}_2)$.

Now let $G_1 = (\Phi_1, \Sigma_1, X_1, P_1, S)$ be an OI-macro grammar that generates L_2. Let $G_2 = (\Phi_2, \Sigma_2, X_1, P_2, S)$ be the OI-macro grammar with $\Phi_2 = \Phi_1 \cup \Sigma_1$ and $P_2 = P_1 \cup \{a \to h(a) \mid a \in \Sigma_1\}$. Let the production π_a be equal to $a \to h(a)$ for each a in Σ_1. The homomorphism $h_3 : P_2^* \to \Sigma_1^*$ is defined by $h_3(\pi) = \lambda$ for $\pi \in P_1$ and $h_3(\pi_a) = a$ for a in Σ_1. Then $h_3^{-1} : \Sigma_1^* \to 2^{P_2^*}$ is a λ-free regular substitution, $h_3^{-1}(L_1^R)$ is in K and $L_{r,\mathrm{OI}}(G_2, h_3^{-1}(L_1^R)) = h(L_1 \cap L_2)$. Hence a language of the form $h(L_1 \cap L_2)$, with L_1 in K and L_2 an OI-macro language, can be generated by some K-controlled (r, OI)-macro grammar (G, C). □

Corollary 4.12. *The family $OI_r(REG)$ equals the family OI.*

Proof. Recall that the family *OI* is closed under intersection with regular sets and under homomorphism [Fis68a]. □

Theorem 4.13. *The language families $RBLB_{r,f,\mathrm{OI}}(OI)$ and $BLB_{r,f,\mathrm{OI}}(OI)$ are equal to the language family OI.*

Proof. Lemma 4.1(iii), Corollaries 4.5 and 4.12, Propositions 4.4 and 4.6. □

Corollary 4.14. $RBLB_{r,f,\mathrm{OI}}(\emptyset NE) = RBLB_{r,f,\mathrm{OI}}(OI) = OI$.

Proof. Corollary 4.5 and Theorem 4.13. □

Of course, a similar statement holds for any family of languages K which satisfies $\emptyset NE \subseteq K \subseteq OI$.

On the other hand, Corollary 4.14 may also be considered as a closure property of the family *OI*, viz., $RBLB_{r,f,\mathrm{OI}}(OI) \subseteq OI$. Whether this property is stronger or weaker than the one established in [Dow] for the family *OI* remains open.

Now we show that the family $BLB_{r,f,\mathrm{IO}}(\emptyset NE)$ differs from the family $BLB_{r,f,\mathrm{OI}}(\emptyset NE)$.

Proposition 4.15. *The family $BLB_{r,f,\mathrm{IO}}(\emptyset NE)$ is not equal to the family OI.*

Proof. The language $L_0 = \{1^m (c\, 1^m)^n \mid n = 2^m - 1,\ m \geq 0\}$ is an IO-macro language which is not an OI-macro language [Fis68a]. The language L_0 can be generated by the $(r, f, \mathrm{IO}, REG, \emptyset NE)$-belb grammar (G, C), where $G =$

(Φ,Ψ,Σ,X,P,S) is defined by $\Phi = \{S,D,F,G\}$, $\Psi = \{\psi_i \mid 0 \le i \le 4\}$, $\Sigma = \{1,c\}$, $X = \{x\}$, and P consists of

$$\begin{aligned}
&\pi_0 = S \to F(\psi_0), && \pi_6 = \psi_1(x) \to \varnothing,\\
&\pi_1 = \psi_0 \to \{1\}, && \pi_7 = G(x) \to \psi_3(x),\\
&\pi_2 = F(x) \to G(\psi_1(x)), && \pi_8 = \psi_2(x) \to \{x\,1\},\\
&\pi_3 = F(x) \to G(\psi_4(x)), && \pi_9 = \psi_3(x) \to \{xcx\},\\
&\pi_4 = D(x) \to \psi_1(x), && \pi_{10} = \psi_4(x) \to \{x\}.\\
&\pi_5 = D(x) \to F(\psi_2(x)), &&
\end{aligned}$$

Finally, take as the control language the trivial control language, i.e., $C = (P \cup \bar{P})^*$. That (G,C) generates L_0 can be shown in the following way. First, a term τ_m of the form $G(G(\ldots G(F(1^m))\ldots))$ is produced, where τ_m contains exactly $m-1$ symbols G $(m \ge 1)$. This can be performed by a control word $\pi_0\pi_1(\pi_2\bar{\pi}_4\pi_5\pi_8)^{m-1}$. Then applying $\pi_3\pi_{10}$, followed by repeatedly applying $\pi_7\pi_9$ yields the string $1^m(c\,1^m)^{2^m-1}$. □

Proposition 4.15 shows a typical consequence of bidirectional rewriting. In case of unidirectional rewriting we have that with $K = \varnothing NE$ for both modes OI and IO, linear basic grammars have the same generating power; viz. they both generate the linear basic languages. The latter equality can also be expressed in terms of (m,K)-elb languages, i.e., let LB denote the family of linear basic languages. Then we have $LB_{\mathrm{OI}}(\varnothing NE) = LB_{\mathrm{IO}}(\varnothing NE) = LB$ $(= EDTOL,$ [Dow]). Due to the presence of bidirectional rewriting in (m,REG,K)-belb grammars we have that the family $BLB_{r,f,\mathrm{OI}}(\varnothing NE)$ differs from the family $BLB_{r,f,\mathrm{IO}}(\varnothing NE)$.

5. Free Rewriting of Nonterminals and Language Names

In this section we study another grammatical model that can be derived from (m,REG,K)-belb grammars. It is natural to investigate also (m,REG,K)-belb grammars provided with the (usual) derivation relation which models the free application of rules from the grammar; i.e., the restriction of right-most rewriting will be dropped in this section. We maintain the restriction of disallowing terminal reductions. Then we prove that the corresponding language family $RBLB_{f,m}(K)$ equals the family of recursively enumerable languages for $m = \mathrm{IO}$ (Proposition 5.3) and for $m = \mathrm{OI}$ (Proposition 5.4), provided some minor conditions on the family K hold.

Definition 5.1. Let $(G,C,\not\subset)$ be an (m,REG,K)-belb grammar, where $G = (\Phi,\Psi,\Sigma,X,P,S)$. Let ρ be rule from $P \cup \bar{P}$, and σ, τ be terms in $Term(G,\not\subset)$. We write $\sigma \Rightarrow_m^{\rho} \tau$ if

- either ρ is a production, σ is rewritten by ρ, and $\sigma \Rightarrow_m \tau$,

- or ρ is a reduction, τ is rewritten by $\bar{\rho}$, and $\tau \Rightarrow_m \sigma$.

In addition, let c be a control word in $C \subseteq (P \cup \bar{P})^*$, with $c = \rho_1 \ldots \rho_n$ ($n \geq 0$, $\rho_i \in P \cup \bar{P}$, $0 \leq i \leq n$). Then $\Rightarrow_m^c$ is defined (as usual) for terms τ and τ' from $Term(G, \Cent)$ by $\tau \Rightarrow_m^c \tau'$ if there are terms τ_i ($0 \leq i \leq n$) such that $\tau_0 = \tau$, $\tau_n = \tau'$ and for each i ($0 \leq i < n$), $\tau_i \Rightarrow_m^{\rho_i} \tau_{i+1}$ holds.

In case a rule ρ_i in c is not applicable to the term τ_i, then further application of rules is blocked, and the application of c to τ yields no result, i.e., there is no term τ' such that $\tau \Rightarrow_m^c \tau'$ is defined. □

In this section an (m,REG,K)-belb grammar will be provided with the derivation relation introduced in Definition 5.1.

Definition 5.2. If $(G,C,\Cent)$ is an (m,REG,K)-belb grammar where $G = (\Phi,\Psi,\Sigma,X,P,S)$ is its underlying grammar, then the *language* generated by $(G,C,\Cent)$ is

$$L_m(G,C,\Cent) = \{w \in \Sigma^* \mid \exists\, c \in C \,.\, S \Rightarrow_m^c w\},$$

where either $m = \mathrm{OI}$ or $m = \mathrm{IO}$. The family of languages generated by (m,REG,K)-belb grammars is denoted by $RBLB_m(K)$. □

As in Section 2 and 3, only (m,REG,K)-belb grammars without terminal reductions will be studied. Such grammars are called (f,m,REG,K)-belb grammars, and their associated family of languages will be denoted by $RBLB_{f,m}(K)$. The next propositions characterize – under minor conditions on the family K – the families $RBLB_{f,\mathrm{IO}}(K)$ and $RBLB_{f,\mathrm{OI}}(K)$; viz. these families equal the family of recursively enumerable languages. The proofs are inspired by the equivalence of Turing machines and on-line acceptors provided with two pushdown stores as auxiliary storage. We refer to Chapter IV, Section 2, for a precise definition of the concept of the Turing-machine as well as related notions and terminology. Given a Turing machine A and a mode m, we construct an (f,m,REG,K)-belb grammar (G,C) which simulates computations of A, using an encoding of two pushdown stores in the derivation tree. These simulations (Propositions 5.3 and 5.4) heavily rely on the use of reductions, which will be no surprise, in view of the main result of Chapter IV.

Remember that *RE* denotes the family of recursively enumerable languages.

Proposition 5.3. *Let K be a family satisfying $\{\{\lambda\},\varnothing\} \subseteq K \subseteq RE$ and let K be closed under left or right-marking. Then a language L_0 is in the family $RBLB_{f,\mathrm{IO}}(K)$ if and only if L_0 is recursively enumerable.*

Proof. Let L_0 be equal to $T(A)$, the set of strings in Σ^* accepted by the deterministic single-tape Turing machine A, where $A = (Q,\Sigma,\Gamma,B,\delta,q_0,F)$.

Furthermore, we assume that $\delta(q,a) = \varnothing$ for each q in F. We construct an (f, IO,*REG*, $\varnothing$*NE*)-belb grammar (G,C) with $G = (\Phi,\Psi,\Sigma,X,P,S)$ such that $L_{\mathrm{IO}}(G,C) = L_0$. The grammar (G,C) generates nondeterministically a representation of a word z in Σ^* and simulates the computation of A on z. The Turing machine A reaches a final state with z as its input if and only if (G,C) generates z. Let Σ_λ denote $\Sigma \cup \{\lambda\}$.

We define the alphabets Φ and Ψ of G by

$$\Phi_0 = \{S, A_0\} \cup \{R_{qDa}, U_{qDa} \mid q \in Q,\ D \in \Gamma,\ a \in \Sigma_\lambda\},$$
$$\Phi_1 = \{[D,a] \mid D \in \Gamma,\ a \in \Sigma_\lambda\},$$
$$\Phi_2 = \{A_1\},$$

and

$$\Psi_0 = \{\psi_1\} \cup \{\xi_q, \eta_q \mid q \in Q\},$$
$$\Psi_1 = \{\psi_a \mid a \in \Sigma_\lambda\},$$
$$\Psi_2 = \{\psi_{A_1}\}.$$

The set of variables X is defined by $X = \{x,y\}$ and the set P of productions by

$$\begin{aligned} P = {} & \{\pi_0, \pi_1, \pi_B, \pi_{q_0}, \pi_{A_1}, \pi_\psi\} \\ & \cup \{U_{pDa} \to \eta_q \mid p,q \in Q,\ D \in \Gamma, a \in \Sigma_\lambda\} \\ & \cup \{R_{pDa} \to \xi_q \mid p,q \in Q,\ D \in \Gamma,\ a \in \Sigma_\lambda\} \\ & \cup \{U_{pDa} \to [E,a](\eta_q) \mid p,q \in Q,\ D,E \in \Gamma,\ a \in \Sigma_\lambda\} \\ & \cup \{R_{pDa} \to [E,a](\xi_q) \mid p,q \in Q,\ D,E \in \Gamma,\ a \in \Sigma_\lambda\} \\ & \cup \{[D,a](x) \to \psi_a(x) \mid D \in \Gamma,\ a \in \Sigma_\lambda\} \\ & \cup \{\psi_a(x) \to \{xa\} \mid a \in \Sigma_\lambda\} \\ & \cup P_\xi \cup P_\Sigma \cup P_\eta \cup \{\psi_1 \to \varnothing\}, \end{aligned}$$

where

$$\begin{aligned} &\pi_0 = S \to A_1(\xi_{q_0}, \psi_1), && \pi_1 = A_0 \to \psi_1, \\ &\pi_B = A_0 \to [B,\lambda](\psi_1), && \pi_{q_0} = A_0 \to \eta_{q_0}, \\ &\pi_{A_1} = A_1(x,y) \to \psi_{A_1}(x,y), && \pi_\psi = \psi_{A_1}(x,y) \to \{xy\}. \end{aligned}$$

Furthermore, let

$$P_\xi = \{\xi_q \to \{\lambda\} \mid q \in F\},$$

$$P_\eta = \{\eta_q \to \{\lambda\} \mid q \in F\},$$
$$P_\Sigma = \{A_0 \to [a,a](\psi_1) \mid a \in \Sigma\}.$$

The control language C is defined by

$$C = \pi_0\bar{\pi}_1(\pi_B\bar{\pi}_1)^*(P_\Sigma\bar{\pi}_1)^*\pi_{q_0}(E_1 \cup M_0 \cup M_{-1}E_{-1})^* M_L^* P_\eta E_\Sigma^+ P_\xi \pi_{A_1}\pi_\psi.$$

The control language C can be considered to consist of three major parts; viz.

initialization part	$\pi_0\bar{\pi}_1(\pi_B\bar{\pi}_1)^*(P_\Sigma\bar{\pi}_1)^*\pi_{q_0}$,
simulation part	$(E_1 \cup M_0 \cup M_{-1}E_{-1})^*$,
termination part	$M_L^* P_\lambda E_\Sigma^+ P_\xi \pi_{A_1}\pi_\psi$.

Before the actual simulation of the Turing machine A starts, the initialization part generates a sentential form $\alpha_{n,k}$, which has a corresponding c-tree of the form shown in Figure 4, where $a_i \in \Sigma$ $(1 \le i \le n)$. If A accepts the string $a_1 \ldots a_n$, then it will stop after some finite computation. The number k is a guess of the number of additional cells to the right of the n input cells, which the Turing machine A uses during this computation.

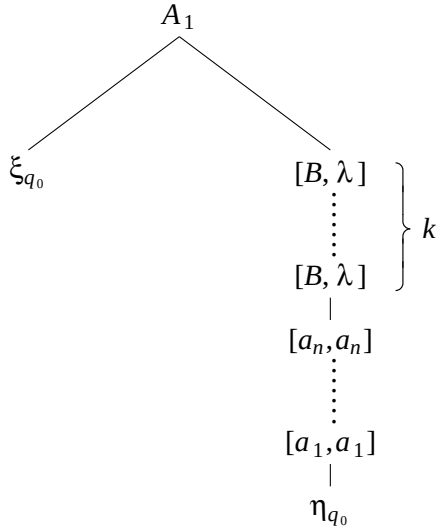

Figure 4.

The simulation part $(E_1 \cup M_0 \cup M_{-1}E_{-1})^*$ simulates the computation of the Turing machine A. The sets E_1, E_{-1}, M_0 and M_{-1} in the simulation part of C are defined as follows. Let $\Delta(r,q,D,a)$ be an abbreviation of

$$r \in \{-1,0,1\},\ q \in Q,\ D \in \Gamma,\ a \in \Sigma_\lambda,\ \exists E \in \Gamma \bullet \exists p \in Q \bullet \delta(q,D) = (p,E,r),$$

and let $[(q,E,r)]_1 = q$ and $[(q,E,r)]_2 = E$. Then we define

$$E_1 = \{(\xi_q \to R_{qDa})([D,a](\eta_q) \to U_{qDa})(R_{qDa} \to [[\delta(q,D)]_2,a](\xi_{[\delta(q,D)]_1}))$$
$$(U_{qDa} \to \eta_{[\delta(q,D)]_1}) \mid \Delta(1,q,D,a)\}.$$

The set E_1 simulates a 1-step of the Turing machine A. The first rule $\xi_q \to R_{qDa}$ of each control string in E_1 is such that D and a are guessed non-deterministically. By the second rule $([D,a](\xi_q) \to U_{qDa})$ this guess is checked, and if the guess happens to be wrong the derivation is blocked.

$$M_s = \{([D,a](\eta_q) \to U_{qDa})(U_{qDa} \to [[\delta(q,D)]_2,a](\eta_{[\delta(q,D)]_1})) \mid \Delta(s,q,D,a)\},$$

where s equals -1 or 0.

The set M_0 simulates a 0-step of the Turing machine A.

$$E_{-1} = \{(\eta_q \to U_{qDa})([D,a](\xi_p) \to R_{qDa})(U_{qDa} \to [D,a](\eta_q))(R_{qDa} \to \xi_q) \mid$$
$$p,q \in Q,\ D \in \Gamma,\ a \in \Sigma_\lambda\}.$$

The sequence $M_{-1}E_{-1}$ simulates a (-1)-step of the Turing machine A.

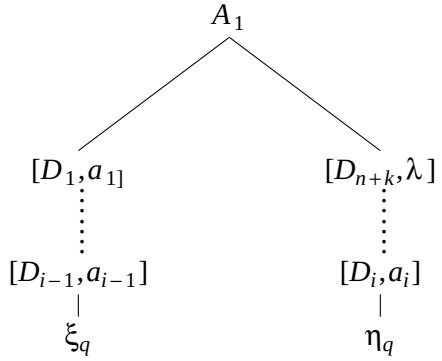

Figure 5.

We can show by induction on the number of Turing machine moves that if

$$q_0 a_1 \ldots a_n \vdash_A^* D_1 \ldots D_{i-1} q\, D_i \ldots D_{n+k},$$

then for some control string c in the simulation part of C we have

$$\alpha_{n,k} \Rightarrow_{\mathrm{IO}}^{c} \omega_{i,q}^{n+k},$$

where $\alpha_{n,k}$ is the sentential form generated by the initialization part of C, corresponding to $q_0 a_1 \ldots a_n$ (cf. Figure 4) and $\omega_{i,q}^{n+k}$ is the sentential form associated with the c-tree represented in Figure 5, where $a_i = \lambda$ for all i with $n+1 \le i \le n+k$, and $D_i \in \Gamma$ for all i with $1 \le i \le n+k$.

If q is in F, then no rules in the simulation part of C are applicable.

In the termination part, sequences from M_L^* – with M_L defined by

$$M_L = \{(\eta_q \to U_{qDa})([D,a](\xi_q) \to R_{qDa})(U_{qDa} \to [D,a](\eta_q))(R_{qDa} \to \xi_q) \mid q \in F,\ D \in \Gamma,\ a \in \Sigma_\lambda\},$$

– transform the sentential form that has been derived after the actual simulation of the Turing machine A into one with a corresponding c-tree of the form shown in Figure 6, where $q \in F$, and $D_i \in \Gamma$ $(1 \le i \le n+k)$.

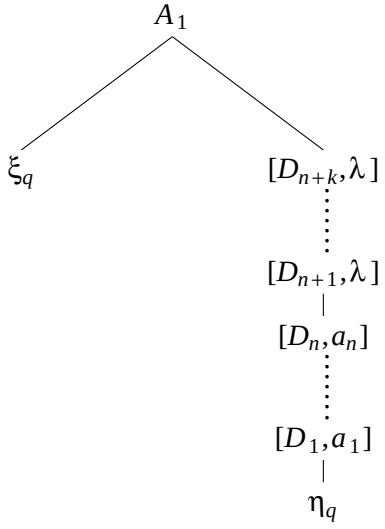

Figure 6.

Finally, $P_\eta E_\Sigma^+ P_\xi \pi_{A_1} \pi_\psi$ derives the terminal string $a_1 \ldots a_n$, where

$$E_\Sigma = \{([D,a](x) \to \psi_a(x))(\psi_a(x) \to \{xa\}) \mid a \in \Sigma_\lambda,\ D \in \Gamma\}.$$

By this construction, we have $T(A) \subseteq L_m(G,C)$. The converse inclusion can also be proved in a straightforward way. Thus, for each Turing machine A we have constructed an (f, IO, REG, K)-belb grammar that generates $T(A)$. This proves the proposition from right to left. The converse implication can be proved using Church's Thesis. □

The construction in the proof of Proposition 5.3 can serve as a base to prove a similar result with respect to the family $RBLB_{f,\,\mathrm{OI}}(K)$.

Proposition 5.4. *Let K be a family satisfying $\{\{\lambda\},\varnothing\} \subseteq K \subseteq RE$, and let K be closed under left or right-marking. Then a language L_0 is an $RBLB_{f,\,\mathrm{OI}}(K)$ language if and only if L_0 is recursively enumerable.*

Proof. We give only the major steps of the construction. The construction of $G = (\Phi, \Psi, \Sigma, X, P, S)$ and C follows the proof of Proposition 5.3 directly. First, the initialization part of C has to generate a sentential form $\alpha_{n,k}$ with a

c-tree structure as shown in Figure 7.

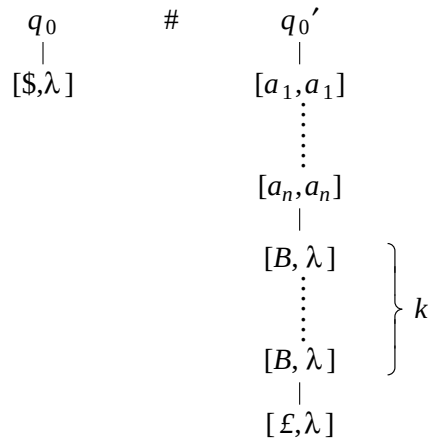

Figure 7.

This can be obtained easily. In general, if $q \in Q$, then q and q' are nonterminals in Φ_1. Nonterminals of the form $[D,a]$, where $D \in \Gamma$ and $a \in \Sigma_\lambda$, are in Ψ_1. Furthermore, $[\$,\lambda]$ and $[\pounds,\lambda]$ are in Ψ_0, where \$, £ are new symbols not in Γ.

The simulation part of C has the form $(E_1 \cup M_0 \cup M_{-1}E_{-1})^*$, which is identical to the simulation part of the control language in the proof of Proposition 5.3. However, the sets E_1, E_{-1}, M_0 and M_{-1} are defined differently. Let R_{qDa}, U_{qDa} ($q \in Q$, $D \in \Gamma$, $a \in \Sigma_\lambda$) and $\Delta(r,q,D,a)$ be defined as in the proof of Proposition 5.3. In addition, we need language names ψ_q, with $q \in Q$. Then

$$E_1 = \{(q(x) \to q([D,a])(x)))\,(q'([D,a](y)) \to U_{qDa}(y))$$
$$(q([D,a](x) \to R_{qDa}(x))\,(R_{qDa}(x) \to [\delta(q,D)]_1([[\delta(q,D)]_2,a](x)))$$
$$(U_{qDa}(y) \to \psi_{[\delta(q,D)]_1}(y))\,(\psi_{[\delta(q,D)]_1}(y) \to [\delta(q,D)]_1'(y)) \mid \Delta(1,q,D,a)\}.$$

To obtain M_0 and $M_{-1}E_{-1}$ we define M_s for s equal to -1 or 0 by

$$M_s = \{(q'([D,a](y)) \to U_{qDa}(y))$$
$$(U_{qDa}(y) \to [\delta(q,D)]_1'([[\delta(q,D)]_2,a](y))) \mid \Delta(s,q,D,a)\}.$$

Finally, E_{-1} is defined by

$$E_{-1} = \{(q'(y) \to q'([D,a](y)))\,(p([D,a](x)) \to R_{qDa}(x))$$
$$(R_{qDa}(x) \to \psi_q(x))\,(\psi_q(x) \to q(x)) \mid p,q \in Q,\ D \in \Gamma,\ a \in \Sigma_\lambda\}.$$

Then it easily follows that a sentential form generated during the simulation of the Turing machine A has a c-tree structure as shown in Figure 8. Note that $a_i = \lambda$ for all i with $n+1 \leq i \leq n+k$.

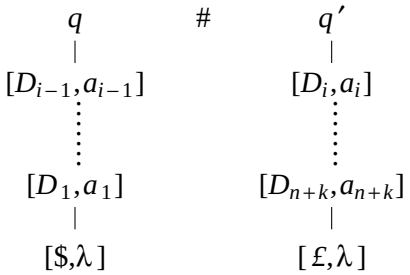

Figure 8.

The discussion of the termination part of C is left to the reader, as well as the remaining details of the proof. □

6. Concluding Remarks

In this chapter we have studied (m,K)-elb grammars provided with regular control and bidirectional rewriting. We have shown that if K is a nontrivial family closed under ngsm mappings, then the family $RBLB_{r,f,\mathrm{OI}}(K)$ is a full substitution-closed AFL. Furthermore, if K is a family with $K \supseteq SYMBOL$, and closed under left or right-marking, intersection with regular languages, and homomorphism, then the family $RBLB_{r,f,\mathrm{IO}}(K)$ is a full Quasi Abstract Family of Languages closed under deterministic substitution.

As for the generating power of these types of grammar we have seen that the family of IO-macro languages is included in each family $RBLB_{r,f,\mathrm{IO}}(K)$, whenever $K \supseteq \varnothing NE$. And similarly, the family of OI-macro languages is included in each family $RBLB_{r,f,\mathrm{OI}}(K)$, whenever $K \supseteq \varnothing NE$. Furthermore, we have that the family $RBLB_{r,f,\mathrm{OI}}(K)$ equals the family OI whenever the family K satisfies $\varnothing NE \subseteq K \subseteq OI$. We also would like to establish upper bounds for the families $RBLB_{r,f,\mathrm{IO}}(\varnothing NE)$. However, the proof techniques applied in the OI-case do not work for the IO-mode, since Lemma 4.10 does not hold for IO-macro grammars.

The results of Section 4 suggest that (m,K)-elb grammars provided with RCB-rewriting generate languages of a "nonlinear" character; i.e., languages generated by a type of grammar provided with symbols similar to nonterminals, such that each grammar, which generates such a language, derives – unidirectionally – at least one sentential form which contains at least two nonterminal-like items. It is likely that this is due to the interaction between the bidirectional rewriting and the presence of language names

nested within nonterminals, which allow to obtain such nonlinear sentential forms. Therefore, it is interesting to study ordinary linear basic grammars provided with RCB-rewriting; cf. Chapter VI.

Finally, with respect to Section 5 we remark that the use of control on the application of rules is indispensable to establish that – under weak assumptions on the family K – the family $RBLB_{f,m}(K)$ equals the family of recursively enumerable languages for both $m = \mathrm{OI}$ and $m = \mathrm{IO}$.

CHAPTER VI

Regularly Controlled Bidirectional Linear Basic Grammars

1. Introduction

In Section V.6 we suggested that it might be interesting to investigate ordinary linear basic grammars as the underlying grammar type for regularly controlled bidirectional grammars. Therefore, in this chapter we extend linear basic grammars to regularly controlled bidirectional linear basic grammars, provided with right-most rewriting, block mode and fair mode.

The structure of this chapter is as follows. In the first part of Section 2 we recall some basic terminology. Then we define a regularly controlled bidirectional linear basic grammar as a tuple $(G,C,\¢)$, where $G = (\Phi,\Sigma,X,P,S)$ is a linear basic grammar, C is a control language over $P \cup \overline{P}$, and $\¢$ is a symbol not occurring in G. The set $\overline{P}$ is formed by the reductions corresponding to the productions in P. In regularly controlled bidirectional linear basic grammars there is – in general – a difference between applying rules from $P \cup \overline{P}$ in the "outside-in" (OI) fashion or in the "inside-out" (IO) fashion. Therefore, we call the regularly controlled bidirectional grammars based on linear basic grammars (m,REG)-blb grammars. Then we actually investigate (m,REG)-blb grammars under the RS/B/f-mode of derivation, where $m = \mathrm{OI}$ or $m = \mathrm{IO}$. The resulting grammar type is denoted by (r,f,m,REG)-blb or even by (f,REG)-blb, since it is argued that in case of the RS/B/f-mode the mode RS does not differ from the RA-mode and even the value of the mode m can be left unspecified.

The remaining part of Section 2 contains some examples of (f,REG)-blb languages. These examples give some insight in the generating power of (f,REG)-blb grammars. Section 3 is devoted to the generating power of (f,REG)-blb grammars. We show that for each recursively enumerable language L_0 over an alphabet Σ_0 there exists an alphabet Σ and some (f,REG)-blb grammar (G,C) such that the language $L(G,C) \cap \Sigma_0^*$ equals L_0. Finally, in Section 4 we draw some conclusions.

2. Regularly Controlled Bidirectional Linear Basic Grammars

Analogously to Chapter V, in which we investigated regularly controlled bidirectional (m,K)-elb grammars, we study in this chapter the effect of regular control together with bidirectional rewriting on linear basic grammars [Fis68a]. Recall that a linear basic grammar is a macro grammar in which the right-hand side of each production is a linear term. In the sequel we assume that each production in a linear basic grammar is in standard linear form; cf. Definition I.2.4.5. For completeness' sake we repeat this definition.

Definition 2.1. A linear basic grammar $G = (\Phi,\Sigma,X,P,S)$ is in *standard linear form* if each production from P has one of the forms

(i) $A(x_1,\ldots,x_n) \to B(w_1,\ldots,w_k)$ or

(ii) $A(x_1,\ldots,x_n) \to w$,

where $w, w_1,\ldots,w_k$ are words over $\Sigma \cup \{x_1,\ldots,x_n\}$. □

For each linear basic grammar we can construct effectively an equivalent linear basic grammar in linear standard form [Fis68a]. Note that this result has already been quoted in Theorem I.2.4.6.

The new grammar model under consideration now consists of a linear basic grammar provided with a control language over $P \cup \overline{P}$. The set $\overline{P}$ consists of the *reductions* corresponding to P. If π is a production in P equal to $A(x_1,\ldots,x_n) \to t$, then $\overline{\pi}$ is in $\overline{P}$ and $\overline{\pi}$ equals $t \to A(\gamma_1,\ldots,\gamma_n)$. Here γ_i is equal to x_i if x_i occurs in t, and otherwise γ_i is equal to $\not\subset$; cf. Chapter V. Furthermore, for each production π we define $\overline{\overline{\pi}}$ equal to π. An element of $P \cup \overline{P}$ will be called a *rule*. The symbol $\not\subset$ is of a special kind and is not an element of Σ, Φ or X.

Definition 2.2. An *m-regularly controlled bidirectional linear basic grammar* or (m,REG)*-blb grammar*, where m is equal to either OI or IO, is a triple $(G,C,\not\subset)$ where

- G is a linear basic grammar (Φ,Σ,X,P,S),
- C is a regular language with $C \subseteq (P \cup \overline{P})^*$,
- $\not\subset$ is a special symbol not occurring in Φ, Σ or X.

We call G the *underlying grammar* of $(G,C,\not\subset)$ and C is called the *control language* of $(G,C,\not\subset)$. Sentences of C will be referred to as *control words*. □

The notion of argument preserving grammars – originally introduced by Fischer in [Fis68a] – has already been defined formally in Chapter V (Definition V.2.4). Obviously, this definition also applies to linear basic grammars. In addition, we need the following concept.

Definition 2.3. A linear basic grammar $G = (\Phi,\Sigma,X,P,S)$ is called *semi-argument preserving* if each production in P of the form 2.1(i) is argument preserving. □

The symbol ¢ can be omitted from the tuple $(G,C,¢)$ in case each production of G is argument preserving.

For an (m,REG)-blb grammar $(G,C,¢)$, with $G = (\Phi,\Sigma,X,P,S)$, let $Term(G,¢)$ denote the set of terms $T(\Sigma \cup X \cup \Phi \cup \{¢\})$. With each (m,REG)-blb grammar we associate the following derivation relation, which formalizes bidirectional right-most rewriting; cf. Definition 2.5.

Our approach is similar to the one we developed in Chapter V. Therefore the following definitions and comments are anything but a surprise.

Definition 2.4. Let $(G,C,¢)$ be an (m,REG)-blb grammar, where G equals (Φ,Σ,X,P,S). Let ρ be a rule from $P \cup \overline{P}$, where $\alpha[\vec{x}]$ is the left-hand side of ρ, and let τ be a term in $Term(G,¢)$. We say that τ *fits in with* ρ, if there are arguments $t_1,\ldots,t_n$ from $Term(G,¢)$ such that $\tau = \alpha[t_1,\ldots,t_n]$, where $\alpha[t_1,\ldots,t_n]$ is the result obtained from $\alpha[\vec{x}]$ by substituting the terms $t_1,\ldots,t_n$ for $x_1,\ldots,x_n$ in $\alpha[\vec{x}]$, respectively. □

Definition 2.5. Let $(G,C,¢)$ be an (m,REG)-blb grammar, where $G = (\Phi,\Sigma,X,P,S)$. Let ρ be rule from $P \cup \overline{P}$, and σ, τ be terms in $Term(G,¢)$. We write $\sigma \Rightarrow^{\rho}_{r,m} \tau$ if there exists a term u in $Term(G,¢)$, and strings v, x and y over the alphabet $\Phi \cup \Sigma \cup X \cup PC$ such that $\sigma = xuy$ and $\tau = xvy$ and

- y contains no symbol from Φ,
- if $u = \lambda$, then $y = \lambda$,
- u is the only subterm in uy that fits in with ρ,
- either ρ is a production, τ is the result of rewriting σ by ρ, and $\sigma \Rightarrow_m \tau$, or ρ is a reduction, σ is the result of rewriting τ by $\overline{\rho}$, and $\tau \Rightarrow_m \sigma$. □

The relation $\Rightarrow^{c}_{r,m}$, where c is a control word in $(P \cup \overline{P})^*$, can be defined in a straightforward way; cf. Definition V.2.8. An (m,REG)-blb grammar provided with right-most rewriting will be called an (r,m,REG)-*blb grammar* or a *right-most regularly controlled bidirectional linear basic grammar*.

Definition 2.6. Let $(G,C,¢)$ be an (r,m,REG)-blb grammar with underlying linear basic grammar $G = (\Phi,\Sigma,X,P,S)$ and control language $C \subseteq (P \cup \overline{P})^*$. Then the language generated by $(G,C,¢)$ under the mode (r,m) is

$$L_{r,m}(G,C,¢) = \{w \in \Sigma^* \mid \exists\, c \in C \,.\, S \Rightarrow^{c}_{r,m} w\}.$$

The family of languages generated by (r,m,REG)-blb grammars is denoted by $RBLB_{r,m}$. □

The derivation relation $\Rightarrow_{r,m}$ defined above corresponds to the RS/B-mode of derivation as defined in Chapter I for RCB grammars.

It is possible to define reductions associated with terminal productions in the obvious way; cf. Definition 2.5. However, we do not study grammatical models in which such general reductions occur. Terminal reductions have the effect that they allow terminals to act as some kind of nonterminal symbol, which obscures the sharp distinction between terminals and nonterminals. We have already noticed this phenomenon several times; cf. Chapter I, II, and V. Restricting ourselves – once again – to non-terminal reductions means that we only consider the *fair mode* of bidirectional rewriting. So in this chapter we also disallow terminal reductions.

It is easy to see that each nonterminal sentential form generated by an (r,m,REG)-blb grammar in fair mode – which we will call an (r,f,m,REG)-*blb grammar* – has a form $uA(\vec{v})w$, where u, w, $v_1,\ldots,v_n$ are strings over the terminal alphabet extended with the symbol ¢. In other words, it is impossible to obtain nested terms. So, the distinction between OI and IO-mode vanishes, and therefore the symbol m can be omitted in the name of grammar and of the language family too. We also see that at most one nonterminal symbol can occur in a sentential form generated by such an (r,f,m,REG)-blb grammar. Thus the symbol r can also be omitted. Therefore we call an (r,f,m,REG)-blb grammar an (f,REG)-*blb grammar* for short. In addition, the language generated by an (f,REG)-blb grammar $(G,C,¢)$ is denoted by $L_f(G,C,¢)$, and the family of languages generated by (f,REG)-blb grammars is denoted by $RBLB_f$. As another consequence, it is easy to see that the following proposition holds.

Proposition 2.7. *For each* (f,REG)*-blb grammar* $(G_0,C_0,¢)$ *there exists an equivalent* (f,REG)*-blb grammar* $(G,C,¢)$ *such that* G *is in standard linear form.* □

Notice that in a semi-argument-preserving (f,REG)-blb grammar $(G,C,¢)$ the symbol ¢ is useless. So we will omit this symbol in semi-argument-preserving (f,REG)-blb grammars.

Example 2.8. Let L_1 be the language $\{1^m(c\,1^m)^n \mid m\geq 1,\ n=2^m-1\}$. In [Fis68a] it has been shown that this language can be generated by an IO-macro grammar, but not by an OI-macro grammar. However, the language L_1 can be generated by the following argument-preserving (f,REG)-blb grammar (G_1,C_1), with $G_1=(\{S,A,B\},\{1,c\},\{x,y\},P,S)$, where the set of productions P consists of

$$\pi_0 = S \to A(c\,1), \qquad \pi_5 = A(x) \to A(1x),$$
$$\pi_1 = A(x) \to A(cx\,1), \qquad \pi_6 = A(x) \to 1x,$$
$$\pi_2 = A(x) \to A(ccx), \qquad \pi_7 = A(x) \to A(c\,1x),$$

$$\pi_3 = B(x,y) \to A(xcc\,1y), \qquad \pi_8 = A(x) \to A(1xc\,1x).$$
$$\pi_4 = B(x,y) \to A(xc\,1yc\,1y),$$

The rank of the symbols in Φ can be easily inferred from the form of the productions in P. Finally, define the control language C_1 by

$$C_1 = \pi_0\pi_1^*(\bar{\pi}_7\pi_8 + \bar{\pi}_2\pi_2\bar{\pi}_3\pi_4)^*\bar{\pi}_5\pi_6.$$

In general, if a production π of the form 2.1(i) in P is argument preserving, we call a sequence $\bar{\pi}\pi$ a *test*. For then we observe that for each string ω to which $\bar{\pi}$ is applicable we have that $\omega \Rightarrow^{\bar{\pi}\pi} \omega$, and if $\bar{\pi}$ is not applicable, then the derivation is blocked by definition. So, a test is able to filter out undesired sentential forms.

That $L_1 = L(G_1, C_1)$ is now shown as follows. First, a sentential form $A(c^m 1^m)$ $(m \geq 1)$ is generated by $\pi_0\pi_1^*$, followed by the test $\bar{\pi}_2\pi_2$ whether the argument of A starts with at least two symbols c. If this is confirmed, the argument string s is split by $\bar{\pi}_3$ into three substrings u, $cc\,1$ and v with $u \in \{c\}^*$ and $s = ucc\,1v$. The next step is to construct from the strings u and v, the string $uc\,1vc\,1v$, which is performed by π_4. Initially, v equals 1^{m-1}, so that the resulting string $uc\,1vc\,1v$ is of the form $c^k(c\,1^m)^l$, where $0 \leq k < m$ and $l = 2^{m-k-1}$. The sequence $\bar{\pi}_7\pi_8$ manages the case in which $c\,1$ is a prefix of the argument of A, and $\bar{\pi}_5\pi_6$ applies in case the argument of A has a prefix equal to 1. □

Example 2.9. The language L_2 defined by $\{w \in \{0,1\}^* \mid \#_1(w) = 2^n,\ n \geq 0\}$, is known to be an OI-macro language that does not belong to the family *IO* [Fis68a]. The language L_2 can also be generated by the following argument-preserving (f, REG)-blb grammar (G_2, C_2), with the underlying grammar $G_2 = (\{S,A,B\}, \{0,1\}, \{x,y\}, P, S)$, where the set of productions P consists of

$$\pi_0 = S \to A(1), \qquad \pi_3 = B(x,y) \to A(x\,0y),$$
$$\pi_1 = A(x) \to A(xx), \qquad \pi_4 = A(x) \to x.$$
$$\pi_2 = B(x,y) \to A(xy),$$

In this case it is also straightforward to determine the rank of the symbols in Φ from these productions. We define the control language C_2 by $C_2 = \pi_0\pi_1^*(\bar{\pi}_2\pi_3)^*\pi_4$.

It is easy to see that $L(G_2, C_2) = L_2$. Note that the sequence $\bar{\pi}_2\pi_3$ has the effect of inserting a symbol 0 somewhere in the current argument string of A. □

3. Generating Power

In Section 2 we showed that both $RBLB_f \cap IO \neq \emptyset$ and $RBLB_f \cap OI \neq \emptyset$; cf. Examples 2.8 and 2.9. The main result of this chapter – formulated in the following proposition – shows that (f, REG)-blb grammars possess a considerable generating power indeed.

Proposition 3.1. *Let Σ_0 be an alphabet. For each recursively enumerable language L_0 over Σ_0 there exist an alphabet Σ and a semi-argument-preserving (f, REG)-blb grammar (G,C) with $G = (\Phi,\Sigma,X,P,S)$ and $\Sigma_0 \subseteq \Sigma$ such that $L_0 = L(G,C) \cap \Sigma_0^*$.*

Proof. Let $A = (Q, \Sigma_0, \Gamma, B, \delta, q_0, F)$ be a deterministic single-tape Turing machine such that L_0 is equal to $T(A)$. For a precise definition of a Turing machine and related notions we refer to Section IV.2. We assume that $\delta(q,a) = \emptyset$ for each q in F and that each symbol in $\Gamma - \Sigma_0$ occurs at least once in some tape contents which is reachable during the computation on some input $a_1 \ldots a_n$ $(n \geq 0)$. We construct a semi-argument-preserving (f, REG)-blb grammar (G,C) with $G = (\Phi,\Gamma,X,P,S)$ such that $L(G,C)$ contains both all sentences w over Σ_0 with $w \in L_0$ as well as each tape contents of A during the computation on w. Of course, then we obtain the equality $T(A) = L(G,C) \cap \Sigma_0^*$. To this end we take the terminal alphabet of G equal to Γ. Next our concern is to assure that each tape symbol from $\Gamma - \Sigma_0$ will occur at least once in a sentence of $L(G,C)$. This is achieved by deriving each possible tape contents which can occur during some (simulated) computation of the Turing machine A. However, it may happen that some tape contents, represented by τ, wholy consists of terminals from Σ_0. Such a string τ is not necessarily an element of $T(A)$ whenever the state of A is not final. So τ has to be excluded from the sentences generated by (G,C). This is done by testing whether or not such a string includes a tape symbol in $\Gamma - \Sigma_0$. If no symbol from $\Gamma - \Sigma_0$ occurs in τ, then the derivation will be blocked. The sets $P_{E,I}$ and P_E in the construction of (G,C) perform this task in the right way. A derivation in an (f, REG)-blb grammar (G,C) starts with producing nondeterministically a word w in Σ_0^* as both the second and the third argument of a nonterminal U. Then it simulates the computation of A on input w. At each stage of the computation the grammar is able to derive the current tape contents as a terminal string, in case this tape contents contains at least one symbol in $\Gamma - \Sigma_0$. In case this simulated computation of A on input w reaches a final state, then the derivation in (G,C) will yield w as the string it generates.

By $\Phi = V_0 \cup V_1 \cup \{S,U\} \cup Q \cup \{E_D^l, E_D^r \mid D \in \Gamma - \Sigma_0\}$ we define the alphabet Φ of G, where $V_0 = Q \times \Gamma$ and $V_1 = V_0 \times \Gamma$.

The set X is equal to $\{x,y,z,x_1,x_2,y_1,y_2\}$. The set P is the union of the finite sets P_I, P_{Σ_0}, P_{ch}, P_i $(i\in\{-1,0,1\})$, $P_{-1,I}$, $P_{E,I}$, P_E, and P_F. The order of description of these sets follows the way in which (G,C) simulates the operation of the Turing machine A.

The subsets $P_I=\{\pi_0,\pi_B,\pi_1\}$ and P_{Σ_0} consist of productions that initialize the simulation of the Turing machine A. These productions are defined by

$$\pi_0 = S \to U(\lambda,\lambda,\lambda), \qquad \pi_1 = U(x,y,z)\to q_0(x,yB,z),$$
$$\pi_B = U(x,y,z)\to U(x,yB,z),$$

and $P_{\Sigma_0}=\{U(x,y,z)\to U(x,ya,za)\,|\,a\in\Sigma_0\}$.

The following five subsets of P – to be defined below – consist of the productions that are necessary to start a simulation of an r-step of the Turing machine A $(r\in\{-1,0,1\})$.

$$P_{ch}=\{(p,D)(x,y,z)\to p(x,Dy,z)\,|\,p\in Q,\ D\in\Gamma,$$
$$\exists E\in\Gamma,\exists q\in Q,\exists r\in\{0,1\}\,.\,\delta(p,D)=(q,E,r)\},$$

$$P_0=\{(p,D)(x,y,z)\to q(x,Ey,z)\,|\,p,q\in Q,\ D,E\in\Gamma,\delta(p,D)=(q,E,0)\},$$

$$P_1=\{(p,D)(x,y,z)\to q(xE,y,z)\,|\,p,q\in Q,\ D,E\in\Gamma,\delta(p,D)=(q,E,1)\},$$

$$P_{-1,I}=\{((p,D),H)(x,y,z)\to(p,D)(xH,y,z)\,|\,p\in Q,\ D,H\in\Gamma,$$
$$\exists E\in\Gamma,\exists q\in Q\,.\,\delta(p,D)=(q,E,-1)\},$$

$$P_{-1}=\{((p,D),H)(x,y,z)\to q(x,HEy,z)\,|\,p,q\in Q,\ \ D,E,H\in\Gamma,$$
$$\delta(p,D)=(q,E,-1)\}.$$

To derive each tape contents with at least one symbol in $\Gamma-\Sigma_0$, the sets $P_{E,I}$ and P_E are defined as follows.

$$P_{E,I}=\{E^l_D(x_1,x_2,y,z)\to p(x_1Dx_2,y,z),\ E^r_D(x,y_1,y_2,z)\to p(x,y_1Dy_2,z)$$
$$\,|\,p\in Q,\ D\in\Gamma-\Sigma_0\},$$

$$P_E=\{E^l_D(x_1,x_2,y,z)\to x_1Dx_2y,\ E^r_D(x,y_1,y_2,z)\to xy_1Dy_2\,|\,D\in\Gamma-\Sigma_0\}.$$

Note that a reduction in $\bar{P}_{E,I}$ can be applied if and only if the (simulated) tape contents includes at least one symbol in $\Gamma-\Sigma_0$.

Once we reach a final state in the simulation of the Turing machine A, the corresponding production in the set $P_F=\{\pi_p\,|\,\pi_p=p(x,y,z)\to z,\ p\in F\}$ generates the terminal string that has apparently been accepted by (the simulation of) the Turing machine A.

Finally, we define the control language C of (G,C) by

$$C = \{\pi_0\}P_{\Sigma_0}^{*}\{\pi_B^{*}\pi_1\}((\overline{P}_{E,I}P_E \cup \overline{P}_{ch}(P_0 \cup P_1) \cup \overline{P}_{-1,I}P_{-1})(P_F \cup \{\lambda\}))^{*}.$$

The construction described above works as follows. Accepting an input string $a_1 \dots a_n$ means that the Turing machine A halts after a finite number of transitions. Apart from the n cells on which the input has been written, A uses an additional number of cells – say k ($k \geq 0$) – to the right of the input, in order to perform the computation on this input. Now we start a derivation of (G,C) by the consecutive application of n ($n \geq 0$) productions from P_{Σ_0} to $U(\lambda,\lambda,\lambda)$, which in turn has been obtained by the initial production π_0. By applying k times ($k \geq 0$) the production π_B, followed by the production π_1, we obtain the sentential form on which the actual simulation of the Turing machine A will take place. So there exists a control string c_1 in $\{\pi_0\}P_{\Sigma_0}^{*}\{\pi_B^{*}\pi_1\}$ such that $S \Rightarrow^{c_1} q_0(\lambda, a_1 \dots a_n B^k, a_1 \dots a_n)$, where $n+k \geq 1$. The term obtained by this subderivation is denoted by $\alpha_{n,k}$.

Next we can simulate the actions of A by applying rules from $P_{-1,I}$ and P_i ($i = -1,0,1$) to $\alpha_{n,k}$. The current state q of A is represented by the nonterminal q from Φ_3. The (values of the) first and second argument form, when concatenated, the current tape contents, such that the position of the head of A is at the left-most symbol of the second argument. Then $\overline{P}_{ch}P_0$ and $\overline{P}_{ch}P_1$ perform actions of A with no head movement and a movement of the head to the right, respectively. In addition, $\overline{P}_{-1,I}P_{-1}$ simulates an action of A in which the head is moved to the left. Thus there exist control strings $c_0 \in \overline{P}_{ch}P_0$, $c_1 \in \overline{P}_{ch}P_1$, and $c_{-1} \in \overline{P}_{-1,I}P_{-1}$ such that

- $p(x,Dy,z) \Rightarrow^{c_0} q(x,Ey,z)$

for each $p,q \in Q$, $x,y \in \Gamma^{*}$, $z \in \Sigma_0^{*}$ and $D,E \in \Gamma$, such that $\delta(p,D) = (q,E,0)$,

- $p(x,Dy,z) \Rightarrow^{c_1} q(xE,y,z)$

for each $p,q \in Q$, $x,y \in \Gamma^{*}$, $z \in \Sigma_0^{*}$ and $D,E \in \Gamma$, such that $\delta(p,D) = (q,E,1)$,

- $p(xH,Dy,z) \Rightarrow^{c_{-1}} q(x,HEy,z)$

for each $p,q \in Q$, $x,y \in \Gamma^{*}$, $z \in \Sigma_0^{*}$ and $D,E,H \in \Gamma$, such that $\delta(p,D) = (q,E,-1)$.

We can show by induction on the number of Turing machine moves that if $q_0 a_1 \dots a_n \vdash_A^{*} b_1 \dots b_{r-1} q\, b_r \dots b_{n+k}$, then for some string c in $(\overline{P}_{ch}(P_0 \cup P_1) \cup \overline{P}_{-1,I}P_{-1})^{*}$ we have

$$\alpha_{n,k} \Rightarrow^{c} q(b_1 \dots b_{r-1}, b_r \dots b_{n+k}, a_1 \dots a_n) \qquad (*)$$

where $b_i \in \Gamma$ ($1 \leq i \leq n+k$). Let the derived string in $(*)$ be denoted by $\omega_{n,k}^{r,q}$.

If a nonterminal symbol q from F appears in $\omega_{n,k}^{r,q}$, then only a rule from P_F is applicable. In that case there exists a production π_q in P_F such that

$$\omega_{n,k}^{r,q} \Rightarrow^{\pi_q} a_1 \ldots a_n.$$

Thus $T(A) \subseteq L(G,C)$. We observe that due to the application of sequences from $\overline{P}_{E,I}P_E$ only tape contents containing at least one symbol from $\Gamma - \Sigma_0$ contribute to $L(G,C)$. So, $T(A)$ includes all strings over Σ_0 in $L(G,C)$. Then it follows that $T(A) = L(G,C) \cap \Sigma_0^*$. This concludes the proof. □

As we have already mentioned before, no nested terms occur in sentential forms derivable by (f, REG)-blb grammars. Therefore, it will be extremely difficult, if not impossible, to find an (f, REG)-blb grammar which can generate $T(A)$. For in that case, the straightforward approach of considering tape symbols in $\Gamma - \Sigma_0$ as nonterminal symbols in the intended (f, REG)-blb grammar causes trouble. This is because the occurrence of two or more tape symbols from $\Gamma - \Sigma_0$ in a tape contents will be hard to represent in such a grammar. Remember that only one nonterminal symbol can occur in sentential forms generated by (f, REG)-blb grammars.

4. Concluding Remarks.

In Chapter V we have seen that the family $RBLB_{r,f,\mathrm{OI}}(\varnothing NE)$ of languages generated by $(r, f, \mathrm{OI}, REG, \varnothing NE)$-belb grammars equals the family OI. Example 2.8 shows that $RBLB_f \neq OI$. So this means that in general regularly controlled bidirectional linear basic grammars have a different generating capacity than regularly controlled bidirectional $(\mathrm{OI}, \varnothing NE)$-elb grammars. This contrasts with the fact that in the unidirectional case we have

$$LB_{\mathrm{OI}}(REG, \varnothing NE) = LB_{\mathrm{OI}}(\varnothing NE) = LB,$$

where $LB_{\mathrm{OI}}(REG, \varnothing NE)$ denotes the family of languages generated by regularly controlled (unidirectional) $(\mathrm{OI}, \varnothing NE)$-elb grammars; cf. [Asv78].

Attempts to prove closure of $RBLB_f$ under concatenation, homomorphism and intersection with regular sets have not been successful. We suppose that this is due to the "stronger" linear character of (f, REG)-blb grammars compared to (m, REG, K)-belb grammars; Section V.6. In this respect Examples 2.8 and 2.9 become even more interesting, as well as Proposition 3.1. So establishing the precise character and expressive power of (m, REG)-blb grammars under the various modes defined in Chapter I, particularly that of (f, REG)-blb grammars, is an obvious but intriguing problem to solve.

CHAPTER VII

Conclusions and Suggestions for Further Research

1. Conclusions

In the preceding chapters we introduced and studied various types of controlled bidirectional grammars. All these types of grammar have been introduced in the following way. Let $\mathcal{G}$ be some grammar type, i.e., a collection of structurally similar grammars. $\mathcal{G}$ may be equal to the family of context-free grammars or the regular grammars to mention but a few concrete examples. Each grammar G of such a type $\mathcal{G}$ possesses a set P of productions. After defining the set $\overline{P}$ of reductions corresponding to the set of productions P, we form a pair (G,C) consisting of a grammar G of type $\mathcal{G}$ and a regular language C over $P \cup \overline{P}$. This construct (G,C) is called a regularly controlled bidirectional $\mathcal{G}$ grammar, or RCB $\mathcal{G}$ grammar. For the following instances of $\mathcal{G}$ we defined the corresponding notion of RCB $\mathcal{G}$ grammar.

$\mathcal{G}$	RCB $\mathcal{G}$
context-free	RCB (context-free)
linear context-free	LRCB
left and right linear context-free	LLRCB, RLRCB
K-extended linear basic	(OI,*REG*,K)-belb, (IO,*REG*,K)-belb
linear basic	(OI,*REG*)-blb, (IO,*REG*)-blb

In addition to the RCB (context-free) grammars we also have defined a time-bounded variant of RCB (context-free) grammars.

Furthermore, we have introduced a collection of modes of derivation m, each of which can be attached to an RCB $\mathcal{G}$ grammar. We have studied RCB (extended) linear basic grammars with respect to one derivation mode, the so-called RS/B/f-mode. The families of languages generated by RCB $\mathcal{G}/m$ grammars have been investigated with respect to closure properties, grammatical transformations (which yield a few normal forms), generating

capacity, and in case of time-bounded RCB (context-free) grammars, also with respect to parsing properties.

Our results give rise to the following concluding observations. First, if we are able to prove a closure property of the family of RCB $\mathcal{G}$ languages in a direct way, then this takes in general much more effort than in the case of the corresponding family of uncontrolled (unidirectional) $\mathcal{G}$ languages. Apart form a usually more complicated construction, due to the presence of reductions, we heavily rely on the control language and the block or skip mode to enforce derivations that possess the desired properties.

It is remarkable that in case of extended linear basic grammars, the families of $RBLB_{r,f,m}(K)$ languages – cf. Chapter V – share so many closure properties with the corresponding families $LB_m(K)$ (where $m = \mathrm{OI}$ or $m = \mathrm{IO}$); cf. [Asv77]. Together with Corollaries V.3.9 and V.3.12 this suggests that (r,f,m,REG,K)-belb grammars inherit many characteristic properties of m-macro grammars. For $m = \mathrm{OI}$ and $\varnothing NE \subseteq K \subseteq OI$ this is confirmed by Corollary 4.14.

Concerning the generating capacity of RCB $\mathcal{G}$ grammars, we observe a considerable increase of generating power, when compared with $\mathcal{G}$ grammars; cf. Chapter IV, Section V.4, and Chapter VI. This is not a complete surprise, although some of the derivation modes lay severe restrictions on the possible derivations in an RCB $\mathcal{G}$ grammar. Remark that the mode RS/B/f gives no increase of generating power in case of RCB (context-free) grammars – compared with (uncontrolled, unidirectional) context-free grammars; cf. Proposition II.2.4(1) – whereas in case of (m,REG,K)-belb grammars ($m = \mathrm{OI}$ or $m = \mathrm{IO}$) it does with respect to (m,K)-elb grammars; cf. Section V.4. We see that the RA and RO-mode of derivation do not decrease the generating power when compared to RCB grammars provided with free application of rules; cf. Chapter IV. Furthermore, an interesting fact is the difference in generating capacity between (f,REG)-blb grammars and $(r,f,\mathrm{OI},REG,\varnothing NE)$-belb grammars, whereas the corresponding unidirectional grammars have equal language generating power; cf. Chapter VI.

2. Suggestions for Further Research.

First of all, we are interested in the position of the families $\mathbf{L}_m$ of RCB/m languages, with $m = \mathrm{RS/B/g}$, $m = \mathrm{RS/S/f}$, and $m = \mathrm{RS/S/g}$ in the Chomsky hierarchy; cf. also Section 2.1 below. The question whether for one of these modes the family $\mathbf{L}_m$ equals the family *CSL* of context-sensitive languages is intriguing. In case the answer is negative, the question can be modified. That is, can we define some new mode of derivation m' such that the family $\mathbf{L}_{m'}$ of RCB/m' languages equals the family *CSL*. Remember that

NSPACE(n) is an alternative characterization of *CSL*; cf. Section IV.6.

From Chapter IV we also recall the open problems whether there exist characterizations for the complexity classes **NP** in terms of polynomial time-bounded RCB/RO grammars, and NTIME(n) in terms of linear time-bounded RCB/RO grammars.

Next, we suggest the investigation of (m,REG,K)-belb grammars provided with a mode which differs from RS/B/f. And the family $RBLB_f$ introduced in Chapter VI has hardly been investigated. In particular, it is very interesting to know whether this family is closed under intersection with regular languages or, at least, under intersection with Σ^* for each alphabet Σ. If this question can be answered in a positive way, Proposition VI.3.1 implies the equality $RBLB_f = RE$ and consequently $RBLB_f$ inherits all (closure) properties from the family *RE*. The proof of Proposition VI.3.1 contains a feature that has its own merit. Viz., a subsequence $\bar{\pi}\pi$ in a control word serves as a partial identity function on strings and so it can be used as a test; cf. Section 2.3 below for an application.

For a number of grammar types $\mathcal{G}$ and modes of derivation m we have seen that the families of RCB/m $\mathcal{G}$ languages equal *RE*. Providing these RCB/m $\mathcal{G}$ grammars with a time bound usually results in a family of recursive languages. In this thesis we only considered time-bounded RCB/m (context-free) grammars in Chapter III. Of course, it is interesting to know the effect of time bounds on the bidirectional grammatical models introduced in Chapters V and VI.

Apart from these questions we discuss in the following two sections two topics of interest in a more detailed way.

2.1. Application of Thue System Theory to RCB Grammars

RCB grammars can be considered as regularly controlled Thue systems together with some kind of restricted application of rules. However, there is another point of view possible from the theory of Thue systems. Therefore we first introduce the following definition. Remember that a linear RCB grammar (G,C) with $G = (V, \Sigma, P, S)$ is in 1-normal form if $V - \Sigma = \{S\}$ and each control word ends with a terminal production; cf. Section II.5.

We call an RCB grammar (G,C) an RLRCB grammar if the underlying grammar G is right-linear.

Definition 2.1.1.

- An LLRCB/f grammar (G,C) is in *strong 1-normal form* if (G,C) is in 1-normal form and the productions of G are of the form $S \to Sa$,

$S \to a$ ($a \in \Sigma$), and $S \to \lambda$.

- An RLRCB/f grammar (G,C) is in *strong 1-normal form* if (G,C) is in 1-normal form and the productions of G are of the form $S \to aS$, $S \to a$ ($a \in \Sigma$), and $S \to \lambda$. □

It is easy to see that the special form of the productions in an LLRCB/f grammar in strong 1-normal form can be obtained in a straightforward way by applying some specific ngsm to an LLRCB/f grammar in 1-normal form; cf. the proof of Proposition II.5.2.

Consider an RCB/f grammar (G,C), where $G = (V, \Sigma, P, S)$ is a regular grammar, i.e., G is either a left-linear or a right-linear context-free grammar. Assume that G is a right-linear context-free grammar. It is easy to see that Proposition II.5.2 holds for RLRCB/f grammars as well. So we assume that (G,C) is in strong 1-normal form, i.e., $V - \Sigma = \{S\}$, P contains only rules of the form $S \to aS$, $S \to a$ ($a \in \Sigma$), and $S \to \lambda$ and for each control word c in C we have that c ends with a terminal production.

In order to prove Proposition 2.1.3 below, we cite the following result concerning monadic Thue systems.

Proposition 2.1.2 [Boo83]. *Let T be a finite monadic Thue system on Σ. For every regular set $D \subseteq \Sigma^*$ the set $\Delta_T^*(D)$ of descendants of D is regular.* □

In addition to Proposition 2.1.2 we note that one can effectively construct from T a finite automaton which accepts the set $\Delta_T^*(D)$; cf. [Boo83].

Proposition 2.1.3. *The family of RLRCB/B/f languages equals the family of regular languages.*

Proof. Let L_0 be an RLRCB/B/f language. Assume that L_0 equals $L(G,C)$, where (G,C) is an RLRCB/B/f grammar in strong 1-normal form. Consider a control word c in C. Then each sequence that consists of a production $\pi_a = S \to aS$ followed by the corresponding reduction $\overline{\pi}_a = aS \to S$ ($a \in \Sigma$) has no net effect to the generation of the ultimately generated terminal string.

If for some control word c in C we have $S \Rightarrow_{B/f}^{c} w$, where w is a terminal string, then each rule in c is applicable. In particular, each reduction in c is applicable. We remark that if at least one reduction occurs in the control word c, then there exists a terminal a such that the sequence $\pi_a \overline{\pi}_a$ does occur in c. So we write c as $c = c_1 \pi_a \overline{\pi}_a c_2$. Then removing the sequence $\pi_a \overline{\pi}_a$ results in a control word $c_1 c_2$ from which it is clear that for $w \in \Sigma^*$, if $S \Rightarrow_{B/f}^{c} w$, then also $S \Rightarrow_{B/f}^{c_1 c_2} w$. Consequently, in $c_1 c_2$ each reduction is applicable too. Thus we can repeatedly apply this process to $c_1 c_2$ until we will end with a control word c' in P^*, such that $S \Rightarrow_{B/f}^{c'} w$ holds.

Let c be some control word that yields no terminal string when applied to S, i.e., the derivation is blocked. Because (G,C) is in strong 1-normal form, this can only be caused by a non-applicable reduction of the form $aS \to S$ $(a \in \Sigma)$ in c. When we repeatedly apply the process of removing sequences of the form $\pi_a \bar{\pi}_a$ $(a \in \Sigma)$ sketched above to such a blocking control word c, sooner or later a sequence of the form $\pi_a \bar{\pi}_b$ has to show up, where a and b are in Σ and $a \neq b$.

Let C' be the control language consisting of all control words c' obtained from control words c from C by removing sequences $\pi_a \bar{\pi}_a$ $(a \in \Sigma)$, such that in c' no sequences of that form occur. Then it will be clear that $L(G,C) = L(G,C')$. Next, we construct the control language C'' from C' by removing each control word from C' in which a reduction occurs. Such control words (in C') cause blocking. Thus $L(G,C'') = L(G,C)$.

The next step of the proof is to show that C'' is regular, from which it follows that $L(G,C)$ is regular [GinSpa]. The cancellation of $\pi_a \bar{\pi}_a$ suggests that Thue systems may be helpful in gaining insight in what kind of control languages the sets C' and C'' are. To this end we introduce the alphabets $\Sigma_0 = \{\tilde{a} \mid a \in \Sigma\}$, $\Sigma_1 = \{\bar{a} \mid a \in \Sigma\}$, $\Sigma_2 = \Sigma \cup \Sigma_0 \cup \Sigma_1$, and $\Sigma_\# = \Sigma_2 \cup \{\#\}$. Note that the alphabets V, Σ_0 and Σ_1 are mutually disjoint. Moreover, # does not occur in $V \cup \Sigma_2$. We define the isomorphism $i : P \cup \bar{P}_f \to \Sigma_\#$ by

$$\begin{aligned} &i(S \to \lambda) = \#, && i(S \to aS) = a, \quad a \in \Sigma, \\ &i(S \to a) = \tilde{a}, \quad a \in \Sigma, && i(aS \to S) = \bar{a}, \quad a \in \Sigma. \end{aligned}$$

As a consequence, for each control word c in C, $i(c)$ ends with a symbol from $\Sigma_0 \cup \{\#\}$. Furthermore, we define the finite special Thue system T over $\Sigma_\#$ by

$$T = \{a\bar{a} \leftrightarrow_T \lambda \mid a \in \Sigma\}.$$

Using the arguments by which we obtained the control language C' from C, discussed above, we see that

$$C' = i^{-1}(\Delta_T^*(i(C)) \cap IRR(T)).$$

Moreover, we can write the control language C'' as

$$C'' = i^{-1}(\Delta_T^*(i(C)) \cap \Sigma^*(\Sigma_0 \cup \{\#\})).$$

Finally, we claim

$$L(G,C) = h(\Delta_T^*(i(C)) \cap \Sigma^*(\Sigma_0 \cup \{\#\})),$$

where $h : \Sigma \cup \Sigma_0 \cup \{\#\} \to \Sigma^*$ is the homomorphism defined by $h(\#) = \lambda$, $h(a) = a$ and $h(\tilde{a}) = a$ $(a \in \Sigma)$.

We note that $IRR(T)$ can be characterized by $\Sigma_{\#}^{*} - (\Sigma_{\#}^{*} dom(T) \Sigma_{\#}^{*})$, thus $IRR(T)$ is a regular set. It follows from Proposition 2.1.2 that C', C'' and $L(G,C)$ are regular sets. □

Remark that C'' can also be defined by a (rather complicated) ngsm mapping T, i.e., $C'' = T(C)$. Since C is regular, so is C''.

It may be interesting to generalize this approach to LRCB/m grammars and RCB/m grammars. This approach may also be fruitful in case of modes different from the RS/B/f-mode. As a promising starting point we use Thue systems in which left-most derivations are defined; cf. [NarOtt] from which we adapt the following terminology and definitions.

Definition 2.1.4. [NarOtt]. Let T be a Thue system on Ξ. The derivation $x \Rightarrow_T y$ is called a *left-most derivation* if there is a rewriting rule (u,v) in T and strings w and z in Ξ^* such that

- $x = wuz$, $y = wvz$, and
- whenever $x = w_1 u_1 z_1$, with $u_1 \in dom(T)$, then
  - wu is a proper prefix of $w_1 u_1$, or
  - $wu = w_1 u_1$, and w is a proper prefix of w_1, or
  - $w = w_1$ and $u = u_1$.

Then $x \Rightarrow_{T,L} y$ denotes that $x \Rightarrow_T y$ is a left-most derivation, and $\Rightarrow_{T,L}^{*}$ denotes the reflexive and transitive closure of $\Rightarrow_{T,L}$. Define for x in Ξ^* the set of left-most descendants of x by

$$\Delta_{T,L}^{*}(x) = \{y \mid x \Rightarrow_{T,L}^{*} y\}.$$

For a language L_0 over Ξ the set of all left-most descendants from words of L_0 is defined by

$$\Delta_{T,L}^{*}(L_0) = \cup\{\Delta_{T,L}^{*}(x) \mid x \in L_0\}. \qquad \square$$

Let R be a rewriting system on Ξ. Then we call x in Ξ^* *r-irreducible* (*modulo R*) if there is no y in Ξ^* such that $x \Rightarrow_R y$. The set $r\text{-}IRR(R)$ denotes the set of r-irreducible words over Ξ by R. A Thue system T on Ξ is called *reduced* if for all $(u,v) \in T$ we have that u and v are in $r\text{-}IRR(T - \{(u,v)\})$, i.e., no rewriting rule can be rewritten on either side by any other rewriting rule of T, when considering T as a rewriting system. As a consequence two different rewriting rules of T have left-hand sides differing from each other [NarOtt]. If a Thue system T is reduced, then for each u which is not r-irreducible (modulo T) – where T is considered as a rewriting system – there exists a unique v in Ξ^* such that $u \Rightarrow_{T,L} v$ [NarOtt].

We can use left-most Thue derivations in the study of LRCB grammars and RCB grammars as follows. Let (G,C) be an LRCB/RS/B/f grammar or

an RCB/RS/B/f grammar, where G is the tuple (V, Σ, P, S). Define the following Thue system T over the alphabet $V \cup \{[,],;\}$ by

$$a[u;v] \leftrightarrow_T [u;v]a, \qquad a \in \Sigma,\ u \to v \in P \cup \bar{P}_f,$$
$$u[u;v] \leftrightarrow_T v, \qquad u \to v \in P \cup \bar{P}_f.$$

For example,

$$baBaa[aBa;A][bA;B] \Rightarrow_{T,L} baBa[aBa;A]a[bA;B]$$
$$\Rightarrow_{T,L} bAa[bA;B] \Rightarrow_{T,L} bA[bA;B]a \Rightarrow_{T,L} Ba.$$

Remark that in the second derivation step the application of $a[aBa;A] \leftrightarrow_T [aBa;A]a$ is not allowed, since we have, following Definition 2.1.4, $wu = w_1 u_1 = baBa$, where $w = b$, $u = aBa[aBa;A]$, and $w_1 = baB$ and $u_1 = a[aBa;A]$.

Note that T is reduced. We see that

$$L_{RS/B/f}(G,C) = \Delta^*_{T,L}(\{S\}h(C)) \cap \Sigma^*, \quad \text{where}$$

$h : P \cup \bar{P}_f \to (V \cup \{[,],;\})^*$ is the isomorphism defined by $h(\rho) = [u;v]$ if $\rho = u \to v$. Note that we actually use T as a rewriting system.

If we can determine what kind of language $\Delta^*_{T,L}(A)$ is – in case A is a K-language, where K is some language family – then we can achieve further results by this approach in which we consider $\Delta^*_{T,L}$ as an operator on language families. In particular the effect of this operator on the family *REG* is one of the first problems to be studied. Furthermore, this method may be modified in order to investigate RCB/m grammars with m equal to RS/B/g, RS/S/f or RS/S/g.

2.2. Fair NTS Grammars

Our source of inspiration to the subject of regularly controlled bidirectional grammars is the concept of NTS grammar; cf. Chapter I. Furthermore, in Chapter IV we showed that the family of RCB/RA/B/f languages equals the family of recursively enumerable languages, even without using control languages or terminal reductions. This observation gives rise to the introduction of fair NTS grammars, formally defined as follows.

Let $G = (V, \Sigma, P, M)$ be a context-free grammar with initial set M. We define the relation $\Rightarrow_{fr}$ on V^* by

$\alpha \Rightarrow_{fr} \beta$ if $\alpha \Rightarrow \beta$ by a production in P_f, where $\alpha, \beta \in V^*$.

Then we define the relation $\Leftrightarrow_{fr}$ on V^* by

$$\alpha \Leftrightarrow_{fr} \beta \quad \text{if} \quad \alpha \Rightarrow \beta \text{ or } \beta \Rightarrow_{fr} \alpha, \qquad \alpha, \beta \in V^*.$$

For each A in $V-\Sigma$ we define $\underline{\underline{LR}}_{fr}(G,A) = \{w \in V^* \mid A \Leftrightarrow_{fr}^* w\}$.

A *fair NTS grammar* is defined as a context-free grammar $G = (V, \Sigma, P, M)$ with initial set M such that for each nonterminal A in $V-\Sigma$, we have

$$\underline{\underline{LR}}_{fr}(G,A) = \underline{L}(G,A).$$

Cf. Section I.2.3 for the definition of $\underline{L}(G,A)$.

A context-free language is a *fair NTS language* if it can be generated by a fair NTS grammar G.

The concepts of NTS grammar and fair NTS grammar differ in the sense that we can find a context-free language which is a fair NTS language, but not an NTS language. Viz., the language

$$L_0 = \{a^n b^n \mid n \geq 1\} \cup \{a^n b^{2n} \mid n \geq 1\}$$

is both not deterministic and not congruential, and therefore not an NTS language; cf. [BoaSen]. The context-free grammar with initial set $\{A,B\}$

$$G = (\{A,B,a,b\}, \{a,b\}, P, \{A,B\}),$$

where P consists of the productions $A \to aAb$, $A \to ab$, $B \to aBbb$, and $B \to abb$, is a fair NTS grammar generating L_0. Moreover, we easily see that each NTS grammar is a fair NTS grammar as well.

We conclude this section with a few questions.

- Do fair NTS grammars possess a “disjoint syntactic category”-like property?
- Does there exist a context-free language that cannot be generated by a fair NTS grammar?

2.3. Some Possible Applications

2.3.1. Relational Databases

Some ideas on RCB $\mathcal{G}$ grammars – developed in this thesis – may possibly be applied in the theory of relational databases, particularly to the problem of query optimization*. For some elementary terminology on relational

* I am indebted to P.M.G. Apers for a discussion on this subject.

databases used in the sequel, we refer to [Ala].

Query optimization is always performed with respect to some cost function which heavily depends on the amount of data that has to be transmitted, especially in case of distributed databases. As an example, consider the (elementary) transformation of the query $\pi_A(R \rtimes_F S)$ into $\pi_{AR}(R) \rtimes_F \pi_{AS}(S)$, where π is the projection operation, and $\rtimes$ is the join operation, R and S are relations, F is a logical condition, and A, AR, AS are sets of attributes. The resulting query has in general a lower cost, for the amount of data to be transmitted in order to be joined together is decreased by first applying the projection operations to the (local) databases. However, this transformation is only permitted in case $Attr(F) \subseteq A$ holds. Here $Attr(\ldots)$ denotes the set of attributes which occur in the argument. The argument F of $Attr(F)$ may be a condition or a relation. Then the sets AR and AS are determined by $AR = A - Attr(S)$, and $AS = A - Attr(R)$. Now it may be interesting to apply the idea of testing as it occurs in RCB $\mathcal{G}$ grammars in order to check whether or not the premise $Attr(F) \subseteq A$ is fulfilled. Remember that "testing" by an RCB $\mathcal{G}$ grammar is performed by a control sequence $\overline{\pi}\pi$ under the block mode. Another problem is whether we can construct the sets AR and AS by means of bidirectional rewriting.

The transformation mentioned above can also be applied in reversed order, under appropriate conditions. Viz., we may transform the query $\pi_{AR}(R) \rtimes_F \pi_{AS}(S)$ into $\pi_A(R \rtimes_F S)$, provided that $A = AR \cup AS$ holds. Notice that in general this transformation increases the cost of the query. Almost all of the elementary query transformations are possible in both directions, under specific conditions. By rearranging a query by such transformations one hopes to achieve a query with minimal cost. It may turn out that, by first applying some transformations which increase the cost of intermediate queries, transformations may become applicable which reduce the cost of the ultimate query below the one of the initial query. This suggests that a bidirectional approach on this level of query optimization may be possible too.

2.3.2. Program Schemes

Another application of RCB $\mathcal{G}$ grammars may be found in the theory of program schemes*; cf. [Eng, Gre75], to which we also refer for unexplained terminology, definitions and notation. Program schemes are obtained from (computer) programs by replacing their instructions by instruction symbols. Thus in program scheme theory we focus our attention to the control

* I thank P.R.J. Asveld for this suggestion.

structure of programs. Each program scheme represents a family of programs. To obtain a program from a program scheme, instruction symbols ought to be interpreted in some way.

As an example, consider the following program scheme Q, informally defined by

$$Q: \quad \textbf{while } p(x) \textbf{ do } x := f(x) \textbf{ od}; \; x := g(x)$$

where p is a partial identity, with $\bar{p}$ as its complement. For instance, let $p(x) = x$ if and only if $x > 0$, let f and g be the predecessor and the successor function, respectively. Under this interpretation with semantic domain equal to $\mathbb{N}$, Q results in a program that computes the constant function $q(x) = 1$ for each $x \in \mathbb{N}$. With each program scheme Q corresponds a language $L(Q)$ called the L-scheme (language viewed as a program scheme; cf. [Eng]) and it equals the set of all possible computations of Q. In case of our example program scheme, $L(Q)$ is equal to $g\bar{p}(fp)^* x$. It is known that if a program scheme Q can be represented by a so-called flow-chart, then $L(Q)$ is a regular language, and if Q is a recursive program scheme, then $L(Q)$ is a context-free language. In the example mentioned above, the symbol x can be considered as a kind of end marker of the sentences in $L(Q)$. If we strip this symbol from the words of $L(Q)$ its structure is still maintained. This also holds for recursive program schemes.

If we allow more than one variable, but still restrict ourselves to unary predicate (or test) symbols, then the corresponding L-schemes are tree languages. For instance, consider the following recursive program scheme U with two variables.

$$x := a; \; y := b; \; S; \; k(x,y)$$

where S is the recursive procedure defined by

$$\begin{aligned} S: \quad & \textbf{if } p(x) \textbf{ then } x := f(x); \; y := g(y); \\ & \qquad\qquad\quad \textit{call } S; \\ & \qquad\qquad\quad y := h(y); \; \textbf{return} \\ & \textbf{else return} \end{aligned}$$

The tree language corresponding to U equals $\{k(\bar{p}(fp)^n a,\, h^n g^n b) \mid n \geq 0\}$.

Note that the symbols a and b are leaves of the trees in $L(U)$. We can consider leaves of a tree as end markers of a tree. This is consistent with strings viewed as monadic (non-branching) trees.

When we allow more general (binary, ternary, ..) predicate symbols, it is hard or even impossible to consider a test like $p(x,y,z)$ as a (unary) partial

indentity. This was one of the reasons why in [Asv78, AsvEng79] nondeterminism in program schemes has been studied, rather than the modeling of (non-unary) tests. A partial solution to this problem may be as follows. We try to transform program schemes into equivalent (controlled) bidirectional extended linear basic (tree) grammars, in which a test $p(x_1, \ldots, x_n)$ is replaced by a (grammatical) test $\bar{\pi}\pi$, where the production π is either of the form I.2.4.5(i) or I.3.3.(i), depending on the respective grammar model. Because $p(x_1, \ldots, x_n)$ is an uninterpreted test, the precise definition of the production π depends on the interpretation which has to be applied to the corresponding program scheme. It cannot be expected that a grammatical test $\bar{\pi}\pi$ can model each possible interpretation of a test $p(x_1, \ldots, x_n)$. Therefore, this approach can only provide a partial solution to this problem.

APPENDIX A

Nonterminal Separating Macro Grammars

1. Introduction

In this Appendix we generalize the NTS (or nonterminal separating) property – originally defined for context-free grammars [Boa]; cf. Section I.2.3 – to macro grammars (Section I.2.4). Then we prove a few characterization results for NTS macro grammars that are analogues of similar results originally established for NTS context-free grammars. We conclude this subject with a few conjectures an an open problem.

In Section 2 we provide the necessary notions, elementary results and terminology on macro grammars and on context-free grammars that satisfy the NTS condition. Section 3 is devoted to the definition of NTS macro grammar and some of their properties as far as they extend the corresponding results on NTS context-free grammars. We restrict our attention to characterization results of the NTS property for *m*-macro grammars where *m* is a mode of derivation, i.e., *m* equals either "outside-in" (or OI), "inside-out" (or IO) or "unrestricted" (or UNR). Finally, Section 4 contains some concluding remarks, open problem, and conjectures.

2. Preliminaries

2.1. UNR-Macro Grammars

Apart from the modes outside-in (OI) and inside-out (IO) (Section I.2.4) we distinguish another mode of derivation for macro grammars; cf. [Fis68a].

In the unrestricted mode (UNR) an occurrence of a nonterminal together with its arguments is expanded according to a production by replacing the nonterminal and its arguments by the right-hand side of that production in which the arguments have been substituted for the corresponding variables.

Definition 2.1.1. Let $G = (\Phi, \Sigma, X, P, S)$ be a macro grammar and let σ and τ be terms over $\Sigma \cup \Phi$. Then we write $\sigma \Rightarrow_{\mathrm{UNR}} \tau$ if

- there is a nonterminal A from Φ_n and terms $\xi_1, \ldots, \xi_n$ over $\Sigma \cup \Phi$ such that $A(\xi_1, \ldots, \xi_n)$ is a subterm of σ;
- $A(x_1, \ldots, x_n) \to t$ is a production from P;
- τ is obtained from σ by replacing the designated term $A(\xi_1, \ldots, \xi_n)$ by t'. The term t' is the result of substituting the terms $\xi_1, \ldots, \xi_n$ for $x_1, \ldots, x_n$ in t, respectively. The term t' is denoted by $t[\xi_1/x_1, \ldots, \xi_n/x_n]$.

The relation $\Rightarrow_{\mathrm{UNR}}$ on $T(\Sigma \cup \Phi)$ represents the UNR-*mode of derivation*, which can be considered as expanding macros without any ordering, or without regarding the depth of nesting of the call. □

Let $\Leftarrow_m$ be the converse of $\Rightarrow_m$, i.e., for all $\sigma, \tau \in T(\Sigma \cup \Phi)$, $\sigma \Leftarrow_m \tau$ holds if and only if $\tau \Rightarrow_m \sigma$. And let $\Leftrightarrow_m$ be the union of $\Rightarrow_m$ and $\Leftarrow_m$. The reflexive and transitive closures of $\Rightarrow_m$, $\Leftarrow_m$ and $\Leftrightarrow_m$ are denoted by $\Rightarrow_m^*$, $\Leftarrow_m^*$ and $\Leftrightarrow_m^*$, respectively. In case $\sigma \Leftarrow_m^* \tau$ [$\sigma \Leftarrow_m \tau$] we say that σ *reduces* [*directly*] *to* τ.

It is easy to see that $\Leftrightarrow_m^*$ is a congruence relation with respect to concatenation. Obviously, it is an equivalence relation and the congruency follows from the observation

$$\sigma \Leftrightarrow_m^* \tau \text{ and } \alpha \Leftrightarrow_m^* \beta \text{ imply } \sigma\alpha \Leftrightarrow_m^* \tau\beta.$$

For $m = \mathrm{UNR}$ this is trivial and in the case of $m = \mathrm{OI}$ or $m = \mathrm{IO}$ it follows from the fact that concatenation does not cause any additional nesting. In the sequel an m-macro grammar will have a finite set M ($M \subseteq \Phi_0$) of initial symbols of rank 0 instead of a single initial symbol; cf. the definition of NTS context-free grammar in Section I.2.3.

Analogously to $m = \mathrm{OI}$ and $m = \mathrm{IO}$ we define the language generated by an UNR-macro grammar as follows.

Definition 2.1.2. The *language* generated by a UNR-macro grammar $G = (\Phi, \Sigma, X, P, M)$ with an initial set M ($M \subseteq \Phi_0$) is defined by

$$L_{\mathrm{UNR}}(G) = \{w \in \Sigma^* \mid \exists S \in M \,.\, \Rightarrow_{\mathrm{UNR}}^* w\}.$$

By *UNR* we denote the family of languages generated by UNR-macro grammars. □

In [Fis68a] Fischer proved the equality *OI* = *UNR*, and so the families *IO* and *UNR* are incomparable.

In the sequel many of our results are restricted to macro grammars which possess the property that every term derived by the macro grammar has a derivation that ultimately yields a string over the terminal alphabet. These macro grammars are called admissible macro grammars [Fis68a].

This property is defined as follows.

Definition 2.1.3. An *m*-macro grammar $G = (\Phi, \Sigma, X, P, M)$ with initial set M ($M \subseteq \Phi_0$) is *admissible* if

- either $\Phi = Z$ and $P = \emptyset$,
- or
  - (i) for each $A \in \Phi$, there exists a sentential form of G in which A occurs,
  - (ii) for each $A \in \Phi_n$ ($n \geq 0$) and each $\sigma_1, ..., \sigma_n \in \Sigma^*$ there exists a string w over Σ such that $A(\sigma_1, ..., \sigma_n) \Rightarrow_m^* w$. □

In [Fis68a] it is shown that for each *m*-macro grammar there exists an equivalent admissible *m*-macro grammar.

Example 2.1.4. Let $L_0 \subseteq \{0,1\}^*$ be the language consisting of those words in which the number of 1's is equal to 2^n for some $n \geq 0$. L_0 is generated by the OI-macro grammar $G = (\Phi, \Sigma, X, P, M)$ with with initial set $M = \{S, A\}$, $\Phi = \Phi_0 \cup \Phi_1$, $\Phi_0 = \{S, A\}$, $\Phi_1 = \{B\}$, $X = \{x\}$, $\Sigma = \{0,1\}$ and P consists of the rules

$$\begin{array}{ll} S \rightarrow B(A), & A \rightarrow 0A, \\ B(x) \rightarrow B(xx), & A \rightarrow A0, \\ B(x) \rightarrow x, & A \rightarrow 1. \end{array}$$

In [Fis68a] it has been shown that L_0 cannot be generated by any IO-macro grammar. Notice that G is admissible. □

2.2. The NTS Property for Context-Free Grammars

NTS or nonterminal separating grammars have been introduced by Boasson [Boa]; cf. Section I.2.3. Remember that a context-free grammar possesses the NTS property if its set of sentential forms is invariant when we apply the productions in both directions, i.e., when we use apart from its productions the corresponding reductions too. We recall the following principal result on NTS grammars.

Proposition 2.2.1. [Boa, BoaSén]. *Let $G = (V, \Sigma, P, M)$ be an NTS grammar. Then for all A and B in $V - \Sigma$, we have either $\underline{L}(G,A) \cap \underline{L}(G,B) = \emptyset$ or $\underline{L}(G,A) = \underline{L}(G,B)$.* □

This property motivates the name of the concept defined in Section I.2.3. However, the converse of Proposition 2.2.1 does not hold; e.g., $\{a^n b^n \mid n \geq 1\} \cup \{a^n b^{2n} \mid n \geq 1\}$ is not an NTS language [BoaSén], but it is easy to show that this language can be generated by a grammar that possesses the Disjoint Syntactic Category property; cf. Section I.2.3.

On the other hand NTS grammars can be characterized in the following way.

Theorem 2.2.2. [BoaSén, Sén85]. *Let $G=(V,\Sigma,P,M)$ be a context-free grammar with initial set M. G has the NTS property if and only if for all $A,B\in V-\Sigma$ and for all $\alpha,\beta,u\in V^*$ the following implication holds.*

$$\text{If } A\Rightarrow^*\alpha u\beta \text{ and } B\Rightarrow^* u, \text{ then } A\Rightarrow^*\alpha B\beta.$$

□

3. The NTS Property for Macro Grammars

3.1. Definitions

We use the following notational conventions. Usually, $(\sigma_1,\ldots,\sigma_n)$ is abbreviated to $(\vec{\sigma}_{(n)})$. The subscript (n) is necessary to distinguish for example $A(\vec{x}_{(n)})$ and $B(\vec{x}_{(k)})$. Only if no confusion is possible we write $\vec{x}$. For $A\in\Phi$, $A(\vec{x})$ is the left-hand side of a production; so $A(\vec{x})=A$ if $A\in\Phi_0$.

Definition 3.1.1. Let $G=(\Phi,\Sigma,X,P,M)$ be an m-macro grammar with initial set M. Then the *language* generated by G is

$$L_m(G,M)=\{w\in\Sigma^*\mid \exists S\in M\,.\,S\Rightarrow_m^* w\},$$

and for each $t\in T(\Sigma\cup X\cup\Phi)$,

$$L_m(G,t)=\{w\in(\Sigma\cup X)^*\mid t\Rightarrow_m^* w\},$$

$$\underline{L}_m(G,t)=\{\omega\in T(\Sigma\cup X\cup\Phi)\mid t\Rightarrow_m^*\omega\},$$

$$\underline{\underline{LR}}_m(G,t)=\{\omega\in T(\Sigma\cup X\cup\Phi)\mid t\Leftrightarrow_m^*\omega\}.$$

□

We are now ready to define the nonterminal separating property for m-macro grammars.

Definition 3.1.2. An m-macro grammar $G=(\Phi,\Sigma,X,P,M)$ with initial set M has the *NTS property* or G is an *NTS m-macro grammar*, if for all $n\geq 0$, for all $A\in\Phi_n$, and for all $\{x_1,\ldots,x_n\}\subseteq X$,

$$\underline{\underline{LR}}_m(G,A(\vec{x}))=\underline{L}_m(G,A(\vec{x})).$$

□

Here we consider the variables $x_1,\ldots,x_n$ as members of a terminal alphabet Σ' with $\Sigma\subseteq\Sigma'$, according to Fischer [Fis68a]; cf. also [EngSchVanL].

Proposition 3.1.3. *Let $G=(\Phi,\Sigma,X,P,M)$ be an NTS m-macro grammar with initial set M. Then for all $n,k\geq 0$, $A\in\Phi_n$, $B\in\Phi_k$, $\{x_1,\ldots,x_n\}\subseteq X$, $\{x_1,\ldots,x_k\}\subseteq X$, either*

$$\underline{L}_m(G,A(\vec{x}_{(n)}))\cap\underline{L}_m(G,B(\vec{x}_{(k)}))=\varnothing$$

or

$$\underline{L}_m(G,A(\vec{x}_{(n)})) = \underline{L}_m(G,B(\vec{x}_{(k)})).$$

Proof. Let ω be an element of $\underline{L}_m(G,A(\vec{x}_{(n)})) \cap \underline{L}_m(G,B(\vec{x}_{(k)}))$. Then $A(\vec{x}_{(n)}) \Rightarrow_m^* \omega$ and $B(\vec{x}_{(k)}) \Rightarrow_m^* \omega$. This implies that we have $A(\vec{x}_{(n)}) \Leftrightarrow_m^* B(\vec{x}_{(k)})$. With the NTS property of G we obtain $A(\vec{x}_{(n)}) \Rightarrow_m^* B(\vec{x}_{(k)})$ and $B(\vec{x}_{(k)}) \Rightarrow_m^* A(\vec{x}_{(n)})$. From this we can conclude that the equality $\underline{L}_m(G,A(\vec{x}_{(n)})) = \underline{L}_m(G,B(\vec{x}_{(k)}))$ holds. □

We see that NTS *m*-macro grammars also share a kind of "disjunct syntactic categories" property (or "nonterminal separating property") as context-free grammars; cf. Proposition 2.2.1.

Example 3.1.4. Consider the linear basic macro grammar $G = (\Phi,\Sigma,X,P,M)$ with $\Phi = \Phi_0 \cup \Phi_3$, $\Phi_0 = \{S\} = M$, $\Phi_3 = \{A\}$, $X = \{x,y,z\}$, $\Sigma = \{a,b,c,[,],\#\}$, and P consists of the productions

$$S \to A(\lambda,\lambda,\lambda)$$
$$A(x,y,z) \to A(ax,by,cz)$$
$$A(x,y,z) \to [x\#y\#z]$$

The language generated by G is $L(G,M) = \{[a^n\#b^n\#c^n] \mid n \geq 0\}$, and $\underline{L}(G,S) = \{S\} \cup \{A(a^n,b^n,c^n) \mid n \geq 0\} \cup L(G)$. Because $A(a^n,b^n,c^n)$, $(n \geq 1)$ only reduces to terms $A(a^k,b^k,c^k)$ with $0 \leq k < n$, and $[a^n\#b^n\#c^n]$ only reduces to $A(a^n,b^n,c^n)$, we have $\underline{L}(G,S) = \underline{LR}(G,S)$. A similar argument for $A(x,y,z)$ yields $\underline{L}(G,A(x,y,z)) = \underline{LR}(G,A(x,y,z))$. Thus G is an NTS macro grammar. □

We see also that in case $\Phi = \Phi_0$ and, consequently, G is a context-free grammar with initial set M, Definition 3.1.2 corresponds to the definition of the NTS property for context-free grammars; cf. Section I.2.3.

3.2. Properties of NTS Macro Grammars

This section is devoted to some results which generalize Theorem 2.2.2 to *m*-macro grammars. To facilitate formulation and proofs we use the following notation.

Definition 3.2.1. Let $G = (\Phi,\Sigma,X,P,M)$ be an *m*-macro grammar with initial set M. Then G has *property* $\Pi(m)$ if for all nonterminals $A \in \Phi_n$, $B \in \Phi_k$, and terms u, $\alpha u \beta$ in $T(\Sigma \cup X \cup \Phi)$, with $\{x_1,\ldots,x_n\} \subseteq X$ and $\vec{\sigma}_{(k)} \in T^k(\Sigma \cup X \cup \Phi)$ the following implication holds.

If $A(\vec{x}_{(n)}) \Rightarrow_m^* \alpha u \beta$ and $B(\vec{\sigma}_{(k)}) \Rightarrow_m^* u$, then $A(\vec{x}_{(n)}) \Rightarrow_m^* \alpha B(\vec{\sigma}_{(k)}) \beta$. □

First, we note that property $\Pi(m)$ is a natural extension of the property mentioned in Theorem 2.2.2 in the sense that if $\Phi = \Phi_0$, i.e., G is context-free, the two properties coincide. To establish Theorem 3.2.3 we need the

following lemma.

Lemma 3.2.2. *Let G be an admissible m-macro grammar. Let ω, ψ be terms from $T(\Sigma \cup X \cup \Phi)$. Then $\omega \Rightarrow_{\mathrm{UNR}} \psi$ implies $\omega \Leftrightarrow^*_{OI} \psi$ as well as $\omega \Leftrightarrow^*_{\mathrm{IO}} \psi$. As a consequence we have $\omega \Rightarrow^*_{\mathrm{UNR}} \psi$ implies $\omega \Leftrightarrow^*_m \psi$ for both $m = \mathrm{OI}$ and $m = \mathrm{IO}$.*

Proof. Let ω be a term $\alpha A(\vec{\sigma})\beta$ with $A \in \Phi_n$ $(n \geq 0)$, $\vec{\sigma} \in T^n(\Sigma \cup X \cup \Phi)$. Furthermore, let $\omega \Rightarrow_{\mathrm{UNR}} \psi$ hold, using the production $A(\vec{x}) \to \delta(\vec{x})$, where $\delta(\vec{x})$ is in $T(\Sigma \cup X \cup \Phi)$, i.e., $\psi = \alpha\delta(\vec{\sigma})\beta$.

Let $m = \mathrm{OI}$. First we have $\alpha A(\vec{\sigma})\beta \Rightarrow^*_{\mathrm{OI}} \alpha' A(\vec{\sigma})\beta'$. This is the string obtained from ω such that every term $A(\vec{\sigma})$ is on top level. Next we derive $\alpha' A(\vec{\sigma})\beta' \Rightarrow^*_{\mathrm{OI}} \alpha'\delta(\vec{\sigma})\beta'$. Now all new occurrences of $\delta(\vec{\sigma})$ are on top level; so we can write

$$\alpha'\delta(\vec{\sigma})\beta' \Leftarrow^*_{\mathrm{OI}} \alpha\delta(\vec{\sigma})\beta.$$

Let $m = \mathrm{IO}$. This is similarly to the case $m = \mathrm{OI}$. We use the derivations $A(\vec{\sigma}) \Rightarrow^*_{\mathrm{IO}} A(\vec{t})$, $A(\vec{t}) \Rightarrow^*_{\mathrm{IO}} \delta(\vec{t})$ and $\delta(\vec{t}) \Leftarrow^*_{\mathrm{IO}} \delta(\vec{\sigma})$, where $\vec{t}$ is in $(\Sigma^*)^n$. □

Theorem 3.2.3. *Let G be an admissible m-macro grammar. Then G is an NTS m-macro grammar if and only if G has property $\Pi(m)$.*

Proof. First we prove the *if*-part. We have to show for G satisfying $\Pi(m)$ that for each $A \in \Phi_n$ $(n \geq 0)$, $\underline{L}_m(G, A(\vec{x})) = \underline{LR}_m(G, A(\vec{x}))$. The inclusion from left to right $(\subseteq)$ is trivial. To establish the converse inclusion $(\supseteq)$, we ought to prove that $A(\vec{x}) \Leftrightarrow^*_m t$ implies $A(\vec{x}) \Rightarrow^*_m t$. This is done by induction on the length of $\Leftrightarrow^*_m$.

Basic step $(p = 0)$. $A(\vec{x}) \Leftrightarrow^0_m t$ implies $A(\vec{x}) \Rightarrow^*_m t$ trivially.

Induction step. As induction hypothesis we take $A(\vec{x}) \Leftrightarrow^p_m t$ implies $A(\vec{x}) \Rightarrow^*_m t$. Consider $A(\vec{x}) \Leftrightarrow^{p+1}_m t$. We distinguish two cases.

Case 1. $A(\vec{x}) \Leftrightarrow^p_m t' \Rightarrow_m t$. Obvious.

Case 2. $A(\vec{x}_{(n)}) \Leftrightarrow^p_m t' \Leftarrow_m t$. Suppose that $t \Rightarrow_m t'$ by the derivation step $B(\vec{\sigma}_{(k)}) \Rightarrow_m u$. Furthermore, let $t = \alpha B(\vec{\sigma}_{(k)})\beta$, $t' = \alpha u \beta$ with terms $\alpha u \beta$, u, $B(\vec{\sigma}_{(k)})$ in $T(\Sigma \cup X \cup \Phi)$. By the induction hypothesis we have $A(\vec{x}_{(n)}) \Rightarrow^*_m t'$. Applying $\Pi(m)$ to $A(\vec{x}_{(n)}) \Rightarrow^*_m \alpha u \beta$ and $B(\vec{\sigma}_{(k)}) \Rightarrow_m u$ we obtain $A(\vec{x}_{(n)}) \Rightarrow^*_m \alpha B(\vec{\sigma}_{(k)})\beta = t$. This completes the induction and the proof of the second inclusion.

To prove the *only if*-part we need the following. Let G be an NTS m-macro grammar. Then for all terms u and $\alpha u \beta$ in $T(\Sigma \cup X \cup \Phi)$, nonterminals B in Φ_k, and vectors of terms $\vec{\sigma}_{(k)}$ in $T^k(\Sigma \cup X \cup \Phi)$,

$$B(\vec{\sigma}_{(k)}) \Rightarrow^*_m u \quad \text{implies} \quad \alpha B(\vec{\sigma}_{(k)})\beta \Leftrightarrow^*_m \alpha u \beta.$$

It is easy to see that for $m = \mathrm{IO}$ and $m = \mathrm{UNR}$ this holds even without G being NTS and with $\Rightarrow_m^*$ instead of $\Leftrightarrow_m^*$. For $m = \mathrm{OI}$ we obtain this implication as follows. If $B(\vec{\sigma}_{(k)}) \Rightarrow_{\mathrm{OI}}^* u$ holds, then $B(\vec{\sigma}_{(k)}) \Rightarrow_{\mathrm{UNR}}^* u$ holds trivially. Thus we have $\alpha B(\vec{\sigma}_{(k)})\beta \Rightarrow_{\mathrm{UNR}}^* \alpha u \beta$ and by Lemma 3.2.2 we obtain $\alpha B(\vec{\sigma}_{(k)})\beta \Leftrightarrow_{\mathrm{OI}}^* \alpha u \beta$. Note that because G is NTS, we now can even prove the stronger implication

$$B(\vec{\sigma}_{(k)}) \Rightarrow_{\mathrm{OI}}^* u \text{ implies } \alpha B(\vec{\sigma}_{(k)})\beta \Rightarrow_{\mathrm{OI}}^* \alpha u \beta.$$

Now, if we have $A(\vec{x}_{(n)}) \Rightarrow_m^* \alpha u \beta$ and $B(\vec{\sigma}_{(k)}) \Rightarrow_m^* u$, then we obtain $A(\vec{x}_{(n)}) \Leftrightarrow_m^* \alpha B(\vec{\sigma}_{(k)})\beta$. Since G is NTS with respect to m, we conclude that $A(\vec{x}_{(n)}) \Rightarrow_m^* \alpha B(\vec{\sigma}_{(k)})\beta$. □

3.3. The Pre-NTS Property for Macro Grammars

Closely connected to the NTS property for context-free grammars is the pre-NTS property [Boa, BoaSén, Sén81]; informally, the pre-NTS property equals the NTS property formulated for terminal strings only. It is still an open problem whether these two properties are equivalent for context-free grammars [Boa, BoaSén, Sén81].

In this section we introduce and study the pre-NTS property for m-macro grammars.

Definition 3.3.1. Let $G = (\Phi, \Sigma, X, P, M)$ be an m-macro grammar with with initial set M ($M \subseteq \Phi_0$). Then G is *pre-NTS* or G *has the pre-NTS property* if for all $A \in \Phi_n$ ($n \geq 0$), and $\{x_1, \ldots, x_n\} \subseteq X$,

$$L_m(G, A(\vec{x})) = LR_m(G, A(\vec{x}))$$

where $LR_m(G, A(\vec{x})) = \underline{LR}_m(G, A(\vec{x})) \cap (\Sigma \cup X)^*$. □

Definition 3.3.2. Let $G = (\Phi, \Sigma, X, P, M)$ be an m-macro grammar with initial set M ($M \subseteq \Phi_0$). Then G has *property* $\pi(m)$ if for all $A \in \Phi_n$ ($n \geq 0$), $B \in \Phi_k$, u', $\alpha u \beta \in (\Sigma \cup X)^*$, $\{x_1, \ldots, x_n\} \subseteq X$, and $\vec{\tau} \in T^k(\Sigma \cup X \cup \Phi)$, the following implication holds.

If $A(\vec{x}) \Rightarrow_m^* \alpha u \beta$, $B(\vec{\tau}) \Rightarrow_m^* u$, and $B(\vec{\tau}) \Rightarrow_m^* u'$,

then $A(\vec{x}) \Rightarrow_m^* \alpha u' \beta$. □

We want to prove the equivalence of Definition 3.3.1 and Definition 3.3.2. It turns out to be the easiest way to do this by introducing a second property $\rho(m)$ which is equivalent to both of them.

Definition 3.3.3. An m-macro grammar $G = (\Phi, \Sigma, X, P, M)$, has *property* $\rho(m)$ if for all nonterminals A in Φ_n ($n \geq 0$), terms t in $T(\Sigma \cup X \cup \Phi)$, and

strings u and u' in $(\Sigma \cup X)^*$ the following implication holds.

$$\text{If } A(\vec{x}) \Rightarrow_m^* u, \quad t \Rightarrow_m^* u, \quad \text{and} \quad t \Rightarrow_m^* u', \quad \text{then} \quad A(\vec{x}) \Rightarrow_m^* u',$$

where $\{x_1, \ldots, x_n\} \subseteq X$. □

Theorem 3.3.4. *Let G be an admissible m-macro grammar. Then the following statements are equivalent.*

(1) *G is pre-NTS with respect to m,*
(2) *G has property $\pi(m)$,*
(3) *G has property $\rho(m)$.*

Proof. (1) $\Rightarrow$ (2). Suppose there exist derivations $B(\vec{\tau}) \Rightarrow_m^* u$, $B(\vec{\tau}) \Rightarrow_m^* u'$ and $A(\vec{x}) \Rightarrow_m^* \alpha u \beta$ for u', $\alpha u \beta \in (\Sigma \cup X)^*$. Because $\alpha u \beta$ is a word over $\Sigma \cup X$ there is no distinction between the three modes of reduction from $\alpha u \beta$. Therefore we have $A(\vec{x}) \Rightarrow_m^* \alpha u \beta \Leftarrow_m^* \alpha B(\vec{\tau}) \beta$. Now in $\alpha B(\vec{\tau}) \beta$, $B(\vec{\tau})$ is on top level, so we continue with $\alpha B(\vec{\tau}) \beta \Rightarrow_m^* \alpha u' \beta$ which is a word over $\Sigma \cup X$. Thus $A(\vec{x}) \Leftrightarrow_m^* \alpha u' \beta$ and, as G is pre-NTS with respect to m, $A(\vec{x}) \Rightarrow_m^* \alpha u' \beta$. Hence G has property $\pi(m)$.

(2) $\Rightarrow$ (3). Let $A(\vec{x}) \Rightarrow_m^* u$, $t \Rightarrow_m^* u$ and $t \Rightarrow_m^* u'$. Obviously, it is possible to write t as a unique sequence of terms, viz. $t = t_1 \ldots t_k$, such that no t_i is a concatenation of two or more terms. It is clear that in expanding some t_i, none of the other terms t_j is affected. So we can write u as $u_1 \ldots u_k$ and u' as $u_1' \ldots u_k'$ with $t_i \Rightarrow_m^* u_i$ and $t_i \Rightarrow_m^* u_i'$, respectively. Now we have for some i $(1 \le i \le k)$ that $A(\vec{x}) \Rightarrow_m^* u_1 \ldots u_i \ldots u_k$, $t_i \Rightarrow_m^* u_i$, $t_i \Rightarrow_m^* u_i'$, and with $\pi(m)$ we obtain $A(\vec{x}) \Rightarrow_m^* u_1 \ldots u_i' \ldots u_k$. We apply this argument to each u_i consecutively, which finally yields $A(\vec{x}) \Rightarrow_m^* u_1' \ldots u_k' = u'$, i.e., we obtain the desired result.

(3) $\Rightarrow$ (1). We have to show $LR_m(G, A(\vec{x})) \subseteq L_m(G, A(\vec{x}))$, which we do by induction on the number of reduction steps in $A(\vec{x}) \Leftrightarrow_m^* w$, with w in $(\Sigma \cup X)^*$. We denote this by $\Leftrightarrow_m^{*n}$ which means that $\alpha \Leftrightarrow_m^{*n} \beta$ holds if and only if $\alpha \Leftrightarrow_m^* \beta$ in which n reduction steps have been used.

Basic step $(n = 0)$. $A(\vec{x}) \Leftrightarrow_m^{*0} w$ directly implies $A(\vec{x}) \Rightarrow_m^* w$.

Induction step. As induction hypothesis we have $A(\vec{x}) \Leftrightarrow_m^{*n} w$ implies $A(\vec{x}) \Rightarrow_m^* w$. Let $A(\vec{x}) \Leftrightarrow_m^{*n+1} w$. To show that $A(\vec{x}) \Rightarrow_m^* w$ we look at the last reduction step in $A(\vec{x}) \Leftrightarrow_m^{*n+1} w$. We write this as

$$A(\vec{x}) \Leftrightarrow_m^{*n} t \Leftarrow_m t' \Rightarrow_m^* w.$$

Because G is admissible there is a word $u \in (\Sigma \cup X)^*$ with $t \Rightarrow_m^* u$. Applying the induction hypothesis we obtain $A(\vec{x}) \Rightarrow_m^* u$, with $t' \Rightarrow_m^* u$, and $t' \Rightarrow_m^* w$ together with property $\rho(m)$ this yields $A(\vec{x}) \Rightarrow_m^* w$. □

4. Concluding Remarks

In the previous section we generalized some characterizations of NTS and pre-NTS context-free grammars to corresponding statements for (pre-) NTS *m*-macro grammars. On the other hand one wants results that are specific for NTS macro grammars in the sense that there is no analogue for context-free grammars. Or, in other words, results that are due to the fact that we deal with macro grammars rather than context-free grammars.

A first example of such a result shows that NTS "reduced macro grammars", i.e., admissible NTS macro grammars with no initial symbols in the right-hand sides of their productions, are argument-preserving.

Recall that an *m*-macro grammar $G = (\Phi, \Sigma, X, P, M)$ with initial set M is called argument-preserving if each production in P is argument preserving. And a production $A(\vec{x}) \to t$ from P is argument-preserving if each variable x_i $(1 \le i \le n)$ occurs at least once in the term t; cf. Definition V.2.4.

Proposition 4.1. *Let $G = (\Phi, \Sigma, X, P, M)$ be an admissible NTS m-macro grammar, with no elements of M occurring in the right-hand side of any production. Then G is argument-preserving.*

Proof. Suppose we have a production $A(x_1, \ldots, x_n) \to t$ where $A \notin \Phi_0$, which is not argument-preserving, say x_i does not occur in t, $1 \le i \le n$. Suppose further that we have obtained a word ω in $T(\Sigma \cup \Phi)$ derived from some S in M on which this rule is applicable. Writing the term ω as $\alpha A(\sigma_1, \ldots, \sigma_n)\beta$ we derive

$$\alpha t[\sigma_1/x_1, \ldots, \sigma_{i-1}/x_{i-1}, \sigma_{i+1}/x_{i+1}, \ldots, \sigma_n/x_n]\beta,$$

where $t[\ldots]$ means that each occurrence of x_i has to be substituted by σ_i $(1 \le i \le n)$. This last term however is, for instance, for some B in M reducible to $\alpha A(\sigma_1, \ldots, \sigma_{i-1}, B, \sigma_{i+1}, \ldots, \sigma_n)\beta$, which we write as $\omega(B)$. So we have $S \Leftrightarrow_m^* \omega(B)$. Since G is NTS, we obtain $S \Rightarrow_m^* \omega(B)$. But no production rule can ever introduce a B from M in a sentential form. Thus we cannot derive such a term $\omega(B)$ from S. □

The following statement is much more interesting. However, we are unable to prove it and therefore we formulate it as

Conjecture 4.2. *Each admissible NTS* IO*-macro grammar generates a basic macro language.* □

The first easy step in proving this conjecture, consists of the following observation.

Lemma 4.3. *Let $G = (\Phi, \Sigma, X, P, M)$ be an admissible NTS* IO*-macro grammar. Then for all $A \in \Phi$,*

$$\underline{L}_{\mathrm{UNR}}(G, A(\vec{x})) = \underline{L}_{\mathrm{IO}}(G, A(\vec{x})).$$

Proof. We only have to show $\underline{L}_{\mathrm{UNR}}(G,A(\vec{x})) \subseteq \underline{L}_{\mathrm{IO}}(G,A(\vec{x}))$, since the converse inclusion is trivial. Let $t \in T(\Sigma \cup X \cup \Phi)$ and $A(\vec{x}) \Rightarrow^*_{\mathrm{UNR}} t$. Then we have by Lemma 3.2.2 $A(\vec{x}) \Leftrightarrow^*_{\mathrm{IO}} t$, and using the fact that G is NTS with respect to IO, we obtain $A(\vec{x}) \Rightarrow^*_{\mathrm{IO}} t$. □

In order to complete the proof of Conjecture 4.2 it is sufficient to establish

Conjecture 4.4. *Let $G = (\Phi,\Sigma,X,P,M)$ be an NTS* IO*-macro grammar that contains a nested production*

$$A(\vec{x}) \to B(t_1,\ldots,t_k) \qquad (*)$$

such that at least on term t_i ($t_i \in T(\Phi \cup \Sigma \cup X)$, $1 \le i \le n$) contains a nonterminal symbol. If the term $t(\vec{x})$ is in $L_{\mathrm{UNR}}(G,B(\vec{x}))$, then in the derivation $A(\vec{x}) \Rightarrow^*_{\mathrm{IO}} t(t_1/x_1,\ldots,t_k/x_k)$ the production $(*)$ has not been applied. □

Remember that macro grammars have been introduced in [Fis68a, Fis68b] as a way to describe context-dependent aspects of the syntax of programming languages. They are an extension of context-free grammars generating, for each mode of derivation, a family of languages in between the families of context-free languages and of context-sensitive languages. Though OI-macro languages are able to describe correctly the declaration and use of program variables, they have the disadvantage of possessing an **NP**-complete membership problem. For IO-macro languages the membership problem is reducible in logarithmic space to the membership problem for context-free languages [Asv81]; so it can be solved deterministically in polynomial time or in space $\log^2 n$. But IO-macro grammars seem to be less suitable for modeling the declaration of program variables.

In Section I.2.3 we already mentioned one of the main results in [BoaSén]; each NTS language can be accepted by a deterministic pushdown automaton. So for context-free grammars the NTS property is a proper restriction with respect to language generating power.

Now the obvious question is whether this holds for macro grammars too. More precisely, is the membership problem for NTS OI-macro or NTS UNR-macro languages still **NP**-complete? In case the answer to this question is negative, what is the complexity of membership problem for these NTS OI-macro languages? This latter question is also interesting in case of NTS IO-macro languages.

References

[Abr] A. Abraham: Some questions on phrase structure grammars I, *Comput. Linguist.* **4** (1965) 61-70.

[Aho] A.V. Aho: Indexed grammars – An extension of context-free grammars, *J. Assoc. Comput. Mach.* **15** (1968) 647-671.

[AhoUll] A.V. Aho & J.D. Ullman: *The Theory of Parsing, Translations and Compiling – Volume I: Parsing* (1972), Prentice-Hall, Englewood Cliffs, NJ.

[Ala] S. Alagić: *Relational Database Technology* (1986), Springer-Verlag, Berlin - Heidelberg - New York.

[Asv77] P.R.J. Asveld: Controlled iteration grammars and full hyper-AFL's, *Inform. and Contr.* **34** (1977) 248-269.

[Asv78] P.R.J. Asveld: *Iterated context-independent rewriting – An algebraic approach to families of languages* (1978), Doctoral Dissertation, University of Twente, Enschede, The Netherlands.

[Asv80] P.R.J. Asveld: Space-bounded complexity classes and iterated deterministic substitution, *Inform. and Contr.* **44** (1980) 282-299.

[Asv81] P.R.J. Asveld: Time and space complexity of inside-out macro languages, *Internat. J. Comput. Math.* **10** (1981) 3-14.

[Asv86] P.R.J. Asveld: Complete symmetry in D2L systems and cellular automata, *Internat. J. Comput. Math.* **19** (1986) 211-223.

[AsvEng77] P.R.J. Asveld & J. Engelfriet: Iterated deterministic substitution, *Acta Inform.* **8** (1977) 285-302.

[AsvEng79] P.R.J. Asveld & J. Engelfriet: Extended linear macro grammars, iteration grammars, and register programs, *Acta Inform.* **11** (1979) 259-285.

[AsvHog] P.R.J. Asveld & J.A. Hogendorp: On the generating power of regularly controlled bidirectional grammars, Memorandum INF-89-68 (1989), Department of Computer Science, University of Twente, Enschede, The Netherlands.

[AsvVanL] P.R.J. Asveld & J. van Leeuwen: Infinite chains of hyper-AFL's, TW-memorandum No. 99 (1975), Twente University of Technology, Enschede, The Netherlands.

[AutBoaSén84] J.M. Autebert, L. Boasson & G. Sénizergues: Langages de parenthèses, langages N.T.S. et homomorphismes inverses, *RAIRO Inform. théor./Theor. Inform.* **18** (1984) 327-344.

[AutBoaSén87] J.M. Autebert, L. Boasson & G. Sénizergues: Groups and NTS languages, *J. Comput. System Sci.* **35** (1987) 243-267.

[BakBoo] B.S. Baker & R.V. Book: Reversal-bounded multipushdown machines, *J. Comput. System Sci.* **8** (1974) 315-332.

[Ber] J. Berstel: Congruences plus que parfaites et langages algébriques, *Séminaire d'Informatique Théorique*, Institut de Programmation (1976-77) 123-147.

[Boa] L. Boasson: Dérivations et réductions dans les grammaires algébriques, *in:* J.W. de Bakker & J. van Leeuwen (Eds.): *7th International Colloquium on Automata, Languages and Programming*, Lect. Notes Comp. Sci. **85** (1980) 109-118, Springer-Verlag, Berlin - Heidelberg - New York.

[BoaSén] L. Boasson & G. Sénizergues: NTS languages are deterministic and congruential, *J. Comput. System Sci.* **31** (1985) 332-342.

[Boo71] R.V. Book: Time-bounded grammars and their languages, *J. Comput. System Sci.* **5** (1971) 397-429.

[Boo78] R.V. Book: Simple representation of certain classes of languages, *J. Assoc. Comput. Mach.* **25** (1978) 23-31.

[Boo81] R.V. Book: NTS grammars and Church-Rosser systems, *Inform. Process. Lett.* **13** (1981) 73-76.

[Boo82] R.V. Book: Confluent and other type of Thue systems, *J. Assoc. Comput. Mach.* **29** (1982) 171-182.

[Boo83] R.V. Book: Decidable sentences of Church-Rosser congruences, *Theor. Comput. Sci.* **24** (1983) 301-312.

[Boo87] R.V. Book: Thue Systems as rewriting systems, *J. Symb. Comp.* **3** (1987) 39-68.

[BooJanWra] R.V. Book, M. Jantzen & C. Wrathall: Monadic Thue systems, *Theor. Comput. Sci.* **19** (1982) 231-251.

[Cho56] N. Chomsky: Three models for the description of languages, *IRE Transactions on Information Theory* **2** (1956) 113-124.

[Cho59] N. Chomsky: On certain formal properties of grammars, *Inform. and Contr.* **2** (1959) 137-167.

[Chot] L. Chottin: Strict deterministic languages and controlled rewriting systems, *in:* H.A. Maurer (Ed.): *6th International Colloquium on Automata, Languages and Programming*, Lect. Notes in Comp. Sci. **71** (1979) 104-117, Springer-Verlag, Berlin - Heidelberg - New York.

[Cul] K. Culik II: A purely homomorphic characterization of recursively enumerable sets, *J. Assoc. Comput. Mach.* **26** (1979) 345-350.

[Dow] P.J. Downey: *Formal languages and recursion schemes*, Ph.D. Thesis TR 16-74 (1974), Center for Research in Computing Technology, Harvard University, Cambridge, Mass.

[DusPar] J. Duske & R. Parchmann: Linear indexed languages, *Theor. Comput. Sci.* **32** (1984) 47-60.

[Ear] J. Earley: An efficient context-free parsing algorithm, *Comm. Assoc. Comput. Mach.* **13** (1970) 94-102.

[EhrRoz] A. Ehrenfeucht & G. Rozenberg: On context-free languages not in EDTOL, *RAIRO Inform. théor./Theor. Inform.* **11** (1977) 273-291.

[Eng] J. Engelfriet: *Simple Program Schemes and Formal Languages*, Lect. Notes Comp. Sci. **20** (1974), Springer-Verlag, Berlin - Heidelberg - New York.

[EngSch] J. Engelfriet, E.M. Schmidt: IO and OI (I), *J. Comput. System Sci.* **15** (1977) 328-353.

[EngSchVanL] J. Engelfriet, E.M. Schmidt & J. van Leeuwen: Stack machines and classes of non-nested macro languages, *J. Assoc. Comput. Mach.* **27** (1980) 96-117.

[EngRoz] J. Engelfriet & G. Rozenberg: Fixed point languages, equality languages, and representation of recursively enumerable languages, *J. Assoc. Comput. Mach.* **27** (1980) 499-518.

[Fis68a] M.J. Fischer: *Grammars with macro-like productions*, Ph.D. Thesis (1968), Harvard University, Cambridge, Mass.

[Fis68b] M.J. Fischer: Grammars with macro-like productions, *Proc. 9th Ann. IEEE Symp. on Switching and Automata Theory* (1968) 131-142.

[Fri] I. Friš: Grammars with partial ordering of the rules, *Inform. and Contr.* **12** (1968) 415-425.

[Fro] Ch. Frougny: Simple deterministic NTS languages, *Inform. Process. Lett.* **12** (1981) 174-178.

[FülVág] Z. Fülöp & S. Vágvölgyi: Congruential tree languages are the same as recognizable tree languages – A proof for a theorem of D. Kozen, *Bull. Europ. Assoc. for Theor. Comp. Sci.* No. 39 (1989) 175-185.

[Gin] S. Ginsburg: *Algebraic and Automata-Theoretic Properties of Formal Languages* (1975), North-Holland, Amsterdam.

[GinRoz] S. Ginsburg & G. Rozenberg: T0L schemes and control sets, *Inform. and Contr.* **27** (1975) 109-125.

[GinSpa] S. Ginsburg & E.H. Spanier: Control sets on grammars, *Math. Systems Theory* **2** (1968) 159-177.

[Gla] A.V. Gladkii: On the complexity of derivations in phrase-structure grammar. *Algebri i Logika Sem.* **3** nr.5-6 (1964) 29-44 (in Russian).

[Gre75] S.A. Greibach: *Theory of Program Structures: Schemes, Semantics, Verification*, Lect. Notes Comp. Sci. **36** (1975), Springer-Verlag, Berlin - Heidelberg - New York.

[Gre77] S.A. Greibach: Control sets on context-free grammar forms, *J. Comput. System Sci.* **15** (1977) 35-98.

[Har] M.A. Harrison: *Introduction to Formal Language Theory* (1978), Addison-Wesley, Reading, Mass.

[Hog87] J.A. Hogendorp: Nonterminal separating macro grammars, *in:* P.R.J. Asveld & A. Nijholt (Eds.): *Essays on Concepts, Formalisms, and Tools* (1987) 77-87, C.W.I. Tract no. 42, Centre for Mathematics and Computer Science, Amsterdam.

[Hog88a] J.A. Hogendorp: Controlled rewriting using productions and reductions, *Proceedings of Computing Science in the Netherlands – 1988* (1988) 479-494.

[Hog88b] J.A. Hogendorp: Time-bounded controlled bidirectional grammars, Memorandum INF-88-60 (1988), Department of Computer Science, University of Twente, Enschede, The Netherlands. To appear in *Internat. J. Comput. Math.*.

[Hog89a] J.A. Hogendorp: Controlled bidirectional grammars, *Internat. J. Comput. Math.* **27** (1989) 159-180.

[Hog89b] J.A. Hogendorp: Regularly controlled bidirectional extended linear basic grammars, Memorandum INF-89-69 (1989), Department of Computer Science, University of Twente, Enschede, The Netherlands.

[Hog90] J.A. Hogendorp: Regularly controlled bidirectional linear basic grammars, Memorandum INF-90-40 (1990), Department of Computer Science, University of Twente, Enschede, The Netherlands.

[HopUll69] J.E. Hopcroft & J.D. Ullman: *Formal Languages and Their Relation to Automata* (1969), Addison-Wesley, Reading, Mass.

[HopUll79] J.E. Hopcroft & J.D. Ullman: *Introduction to Automata Theory, Languages, and Computation* (1979), Addison-Wesley, Reading, Mass.

[Kas] T. Kasai: An hierarchy between context-free and context-sensitive languages, *J. Comput. System Sci.* **4** (1970) 492-508.

[Kha74a] N.A. Khabbaz: A geometric hierarchy of languages, *J. Comput. System Sci.* **8** (1974) 142-157.

[Kha74b] N.A. Khabbaz: Control sets on linear grammars, *Inform. and Contr.* **25** (1974) 206-221.

[Koz] D. Kozen: Complexity of finitely generated algebras, *Proc. 9th Annual ACM Symp. on Theory of Computing* (1977) 164-177.

[Lan] L.C. Langlois: *Parallel parsing of context-free languages on an array of processors*, Ph.D. Thesis (1988), University of Edinburgh, Scotland, UK.

[LewPap] H.R. Lewis & C.H. Papadimitriou: *Elements of the Theory of Computation* (1981), Prentice-Hall, Englewood Cliffs, NJ.

[Mat] G.H. Matthews: A note on asymmetry in phrase structure grammars, *Inform. and Contr.* **7** (1964) 360-365.

[McNNarOtt] R. McNaughton, P. Narendran & F. Otto: Church-Rosser Thue systems and formal languages, *J. Assoc. Comput. Mach.* **35** (1988) 324-344.

[NarOtt] P. Narendran & F. Otto: Some polynomial algorithms for finite monadic Church-Rosser Thue systems, *Theor. Comput. Sci.* **68** (1989) 319-332.

[Nie] M. Nielsen: E0L systems with control devices, *Acta Inform.* **4** (1975) 373-386.

[Niv] M. Nivat: On some families of languages related to the Dyck systems, *Proc. 2nd ACM Sym. Theory Comp.* (1970) 221-225.

[Ros] D.J. Rosenkrantz: Programmed grammars and classes of formal languages, *J. Assoc. Comput. Mach.* **16** (1969) 107-131.

[Sal69] A. Salomaa: On grammars with restricted use of productions, *Ann. Acad. Sci. Fennicae Ser. AI.* **454** (1969).

[Sal70] A. Salomaa: On some families of formal languages obtained by regulated derivations, *Ann. Acad. Sci. Fennicae Ser. AI.* **479** (1970).

[Sal73] A. Salomaa: *Formal Languages* (1973), Academic Press, New York.

[Sav] W.J. Savitch: How to make arbitrary grammars look like context-free grammars, *SIAM J. Comput.* **2** (1973) 174-182.

[Sén81] G. Sénizergues: A new class of C.F.L. for which the equivalence is decidable, *Inform. Process. Lett.* **13** (1981) 30-34.

[Sén85] G. Sénizergues: The equivalence and inclusion problems for NTS languages, *J. Comput. System Sci.* **31** (1985) 303-331.

[Sén89] G. Sénizergues: Church-Rosser controlled rewriting systems and equivalence problems for deterministic context-free languages, *Inform. and Comp.* **81** (1989) 265-279.

[Sud] T.A. Sudkamp: *Languages and Machines – An Introduction to the Theory of Computer Science* (1988), Addison-Wesley, Reading, Mass.

[Thu] A. Thue: Probleme über Veranderungen von Zeichenreihen nach gegeben Regeln, *Skr. Vid. Kristianaia I. Mat. Naturv. Klasse,* **10/34** (1914).

[VanL] J. van Leeuwen: *Rule-labeled programs – A study of a generalization of context-free and some classes of formal languages* (1972), Doctoral Dissertation, University of Utrecht, The Netherlands.

[Vog] H. Vogler: The OI-hierarchy is closed under control, *Inform. and Comp.* **78** (1988) 187-204.

Gearfetting

Yn ’e measte grammatikamodellen befettet in grammatika in samling werskriuwrigels. Dizze werskriuwrigels wurde ien kant út tapast (fan lofts nei rjochts) en wurde ek wol produksjes neamd. Yn tsjinstelling ta soksoarte unidireksje grammatika’s kinne yn de yn dit proefskrift definiearre bidireksje grammatika’s de werskriuwrigels beide kanten út tapast wurde. In werskriuwrigel dy’t tapast wurdt fan rjochts nei lofts wurdt in reduksje neamd. Hjirnei ferstean wy ûnder in rigel sawol in produksje as in reduksje.

It brûken fan werskriuwrigels yn beide rjochtingen hat ûnder frije tapassing fan ’e rigels (produksjes en reduksjes) in tige grutte tanimming fan ’e generative krêft ta gefolch. Om dizze tanimming yn’e hân te hâlden wurdt oan in bidireksje grammatika in bestjoeringsmeganisme op ’e tapassing fan ’e rigels yn’e foarm fan in reguliere bestjoeringstaal oer dizze rigels taheake. Om ’e generative krêft te beheinen wurde alternative wizen fan ôflieden (“modes of derivation”) bestudearre, en wol de rjochter-ôfliedingswize en de rjochterfoarkommen-ôfliedingswize. Boppedat jout it ûnderskieden fan twa soarten fan reduksjes in twadde mooglikheid ta it beheinen fan ’e generative krêft. Dizze twa binne de “suvere” reduksjes en de “algemiene” reduksjes. As lêste jout de oanwêzigens fan bestjoeringstalen oanlieding ta in tredde gefalsûnderskieding. Nammentlik, as in rigel oanjûn troch in wurd út ’e bestjoeringstaal net tapasber is, dan kinne wy as stopje en gjin inkele sinsfoarm ôfleverje (blokkearing), as trochgean mei de folgjende rigel oandroegen troch it bestjoeringswurd (oerslaan). Kombinaasjes fan dizze ôfliedingswizen jouwe acht ferskate gearstâlde ôfliedingswizen.

Boppesteande ôfliedingswizen wurde yn haadstik I en II definiearre. Fierders is haadstik I ynliedend fan karakter, en befettet it in oantal technyske definysjes dy’t fierder yn it proefskrift brûkt wurde sille. Ek wurde yn dit haadstik de grammatikamodellen út ’e haadstikken II oan’ta mei IV yn’t ramt fan de theory fan Thue-systemen brocht.

Yn haadstik II wurde regulier bestjoerde bidireksje grammatika’s basearre op kontekst-frije grammatika’s bestudearre. Der wurde ôfslutingseigenskippen fêststeld en ek wurdt de generative krêft fan ien fan ’e gearstâlde ôfliedingswizen, de saneamde RS/B/f-wize – rjochterôflieding, blokkearing, en suvere reduksjes – bepaald. It docht bliken dat dizze krêft gelyk is oan dy fan kontekst-frije grammatika’s. Foar dizze RS/B/f-ôfliedingswize kinne wy ek in normaalfoarm-theorema bewize. Fierders wurde yn dit haadstik ek noch regulier bestjoerde bidireksje grammatika’s op basis fan (lofts) lineêre kontekst-frije grammatika’s ûndersocht. As lêste generalisearje wy de reguliere bestjoering ta bestjoeringstalen út likefolle wat foar taalfamyljes.

Haadstik III is wijd oan saneamde (ôfliedings)lingte-begrinsde farianten fan de yn haadstik I definiearre grammatika's. Yn dizze lingte-begrinsde regulier bestjoerde bidireksje grammatika's wurde allinnich dy ôfliedingen talitten dy't in yn it foar fêststelde lingte net te boppe gean. Dizze boppegrins hinget allinnich ôf fan 'e lingte fan 'e lang om let te generearjen sin. Foar regulier bestjoerde bidireksje grammatika's is dit wichtich omdat der yn dit type grammatika yn 'e ôfliedingen lutsen sûnder effektive werskriuwing fan (dielen fan) 'e sinsfoarm foarkomme kinne. Foar dizze lingte-begrinsde farianten wurde ôfslutingseigenskippen fan 'e oerienkomstige taalfamyljes fêststeld en wurdt in normaalfoarmstelling bewiisd. Ek binne der foar dit type grammatika ûntleders te konstruerjen dy't – fansels– terminearje foar eltse ynfier.

Haadstik IV giet fierder mei it fêststellen fan 'e generative krêft fan de typen grammatika's definiearre yn haadstik I en II. Foar fjouwer fan 'e gearstâlde ôfliedingswizen komme wy út op de famylje fan rekursyf op te somjen talen. Ien jout krekt de kontekst-frije talen (haadstik II); de oare trije binne krêfticher dan kontekst-frije grammatika's, mar de krekte krêft is noch ûnbekind.

Yn haadstik V wurde regulier bestjoerde bidireksje grammatika's op basis fan saneamde "extended linear basic" grammatika's bestudearre. Dit type grammatika wurdt mei ien fan 'e gearstâlde ôfliedingswizen kombinearre, de RS/B/f-wize. Ofslutingseigenskippen en generative krêft fan de taalfamyljes generearre troch dit type grammatika wurde fêststeld, wat inkele nijsgjirriche risseltaten oplevert. As lêste wurdt ek de generative krêft yngefal fan "frije rigel tapassing" yn sinsfoarmen fêststeld.

Haadstik VI is feitlik in earste oanset ta in ûndersyk fan regulier bestjoerde bidireksje grammatika's op basis fan "linear basic" grammatika's. It docht bliken dat dit nije, relatyf ienfâldiche grammatikamodel in opfallende generatyfe krêft hat.

By einbeslút wurde yn haadstik VII ferskate ynteressante fragen en suggestjes foar fierder ûndersyk formulearre. Ek jouwe wy in twatal gebieten fan 'e theoretyske ynformatika oan weryn faaks mei sukses de yn dit proefskrift ûntwikkelde grammatikamodellen tapast wurde kinne.

Samenvatting

In de meeste grammaticamodellen bevat een grammatica een verzameling herschrijfregels. Deze herschrijfregels worden in één richting toegepast (van links naar rechts) en worden ook wel producties genoemd. In tegenstelling tot dergelijke unidirectionele grammatica's kunnen in de in dit proefschrift gedefinieerde bidirectionele grammatica's de herschrijfregels in beide richtingen toegepast worden. Een herschrijfregel toegepast van rechts naar links wordt een reductie genoemd. Onder een regel verstaan we in het vervolg een productie of een reductie.

Het gebruik van herschrijfregels in beide richtingen veroorzaakt onder vrije toepassing van regels (producties en reducties) een enorme toename in generatieve kracht. Om deze toename in de hand te houden wordt aan een bidirectionele grammatica een besturingsmechanisme op de toepassing van de regels in de vorm van een reguliere besturingstaal over deze regels toegevoegd. Om de generatieve kracht in te perken worden alternatieve wijzen van afleiden ("modes of derivation") bestudeerd, te weten de rechter-afleidingswijze, en de rechtervoorkomen-afleidingswijze. Bovendien levert het onderscheiden van twee soorten van reducties een tweede mogelijkheid op tot het inperken van de generatieve kracht. Deze twee zijn de "zuivere" reducties en de "algemene" reducties. Tenslotte geeft de aanwezigheid van besturingstalen aanleiding tot een derde gevalsonderscheiding. Namelijk, als een regel aangegeven door een woord uit de besturingstaal niet toepasbaar is, kunnen we òf stoppen en geen enkele zinsvorm afleveren (blokkering), òf doorgaan met de volgende regel aangedragen door het besturingswoord (overslaan). Combinaties van deze afleidingswijzen geven acht verschillende samengestelde afleidingswijzen.

Bovenstaande afleidingswijzen worden in hoofdstuk I en II gedefinieerd. Verder is hoofdstuk I inleidend van karakter, en bevat het een aantal technische definities die verder in het proefschrift gebruikt zullen worden. Ook worden in dit hoofdstuk de grammaticamodellen uit de hoofdstukken II tot en met IV in het kader van de theorie van Thue-systemen geplaatst.

In hoofdstuk II worden regulier bestuurde bidirectionele grammatica's gebaseerd op context-vrije grammatica's bestudeerd. Er worden afsluitingseigenschappen bepaald en ook wordt de generatieve kracht van één van de samengestelde afleidingswijzen, de zogenaamde RS/B/f-wijze – rechterafleiding, blokkering, en zuivere reducties – vastgesteld. Deze kracht blijkt gelijk te zijn aan die van context-vrije grammatica's. Voor deze RS/B/f-afleidingswijze kunnen we ook een normaalvorm-theorema bewijzen. Verder

worden in dit hoofdstuk ook nog regulier bestuurde bidirectionele grammatica's op basis van (links) lineaire context-vrije grammatica's onderzocht. Tenslotte generaliseren we de reguliere besturing tot besturingstalen uit willekeurige taalfamilies.

Hoofdstuk III is gewijd aan zogenaamde (afleidings)lengte-begrensde varianten van de in hoofdstuk I gedefinieerde grammatica's. In deze lengte-begrensde regulier bestuurde bidirectionele grammatica's worden slechts die afleidingen toegelaten die een vooraf bepaalde lengte niet te boven gaan. Deze bovengrens hangt slechts af van de lengte van de uiteindelijk te genereren zin. Voor regulier bestuurde bidirectionele grammatica's is dit van belang omdat in dit type grammatica er in de afleidingen lussen zonder effectieve herschrijving van (delen van) de zinsvorm kunnen voorkomen. Voor deze lengte-begrensde varianten worden afsluitingseigenschappen van de overeenkomstige taalfamilies bepaald en wordt een normaalvorm-stelling bewezen. Ook zijn er voor dit type grammatica ontleders te construeren die – vanzelfsprekend – termineren voor elke invoer.

Hoofdstuk IV vervolgt met het bepalen van de generatieve kracht van de typen grammatica's gedefinieerd in hoofdstuk I en II. Voor vier van de samengestelde afleidingswijzen komen we uit op de familie van recursief opsombare talen. Eén levert precies de context-vrije talen op (hoofdstuk II); de overige drie zijn krachtiger dan de context-vrije grammatica's, maar de precieze kracht is nog onbekend.

In hoofdstuk V worden regulier bestuurde bidirectionele grammatica's op basis van zogenaamde "extended linear basic" grammatica's bestudeerd. Dit type grammatica wordt met een van de samengestelde afleidingswijzen, de RS/B/f-wijze, gecombineerd. Afsluitingseigenschappen en generatieve kracht behorend bij dit type grammatica worden bepaald, hetgeen tot enkele opmerkelijke resultaten leidt. Tenslotte wordt ook de generatieve kracht ingeval van "vrije regel toepassing" in zinsvormen bepaald.

Hoofdstuk VI is in feite een eerste aanzet tot een onderzoek van regulier bestuurde bidirectionele grammatica's op basis van "linear basic" grammatica's. Dit nieuwe, betrekkelijk eenvoudige grammaticamodel blijkt een opvallende generatieve kracht te bezitten.

Tenslotte worden in hoofdstuk VII diverse interessante vragen en suggesties voor verder onderzoek geformuleerd. Ook geven we een tweetal gebieden van de (theoretische) informatica aan waarin wellicht met succes de in dit proefschrift ontwikkelde grammaticamodellen kunnen worden toegepast.