Time-Bounded Controlled Bidirectional Grammars

Jan Anne Hogendorp*

*Department of Computer Science, University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands*

Abstract — We study regularly controlled bidirectional (RCB) grammars from the viewpoint of time-bounded grammars. RCB-grammars are context-free grammars of which the rules can be used in a productive and in a reductive fashion, while the application of these rules is controlled by a regular language. Several modes of derivation can be distinguished for this kind of grammar. A time-bound on such a grammar is a measure of its derivational complexity. For some families of time bounds and for some modes of derivation we establish closure properties and a normal form theorem. In addition parsing algorithms are given for some modes of derivation. We conclude with considering generalizations with respect to the family of control languages and the family of bounding functions.

1. Introduction

In [8] we introduced RCB-grammars, i.e., context-free grammars in which the rules can be used in both a productive as in a reductive fashion. The application of these rules is controlled by a regular language C of control words, which are strings consisting of productions and reductions (i.e., productions used in the reversed way). We denote an RCB-grammar by (G, C) where $G = (V, \Sigma, P, S)$ is a context-free grammar, such that $C \subseteq (P \cup \bar{P})^*$, where $\bar{P}$ is the set of reductions $\{\alpha \rightarrow A \mid A \rightarrow \alpha \in P\}$. These grammars have been inspired by the concept of NTS-grammar: NTS or nonterminal separating grammars form a subclass of the context-free grammars [3]; cf. also [2]. In this type of grammar, each sentential form that can be derived from a non-terminal by means of both productions and reductions can also be derived by the use of productions only. Since in RCB-grammars reductions are allowed, there is also a connection with *Thue systems*; cf. [5, 10]. An RCB-grammar can be considered as a particular kind of Thue system with a restricted use (viz. a controlled use) of the rewriting rules. The context-free grammar G of an RCB-grammar (G, C) is referred to as the *underlying* grammar of (G, C) . In combination with this kind of grammar, we distinguish – as in [8,9] – several modes of derivation, which are described in Section 2. For each mode of derivation m , an RCB-grammar (G, C) gives rise to a language $L_m(G, C)$. Therefore we obtain for each mode of derivation a corresponding language family.

The presence of reductions in RCB-grammars allows the construction of a control string c with length $|c|$ greater than zero such that for some string α we have $\alpha \Rightarrow^c \alpha$. If such a control string occurs as c^+ or c^* as part of a

* The work of the author has been supported by the Netherlands Organization for Scientific Research (N.W.O.).

control language, it is very hard to construct a parser for an RCB-grammar that terminates for each input string. It is not at all clear 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 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*. 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 $\phi: \mathbb{N} \rightarrow \mathbb{R}$ 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 [6] to describe the derivational complexity of general phrase-structure grammars. In [4] bounding functions have been used to generate particular language families; thus this paper may also be considered as an extension of [4].

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 [1, 2], (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.

The paper is organized as follows. In Section 2 the definition of RCB-grammar – introduced in [8] – is recalled, including the various modes of derivation. A mode consists of three submodes, each possessing two instances. The first submode provides two alternatives to select the nonterminal to be rewritten. If a rule cannot be applied according the first submode, we have two other possibilities. Either we skip this rule and try to apply the next one, or we abort the derivation, producing no string at all. The third submode makes a distinction between the form of the rules. Either

we allow rules of the form $\alpha \rightarrow A$ with $\alpha \in \Sigma^*$, or we disallow such rules. The first possibility has some links with Thue systems, the second keeps the connection intact with the concept of phrase-structure grammar, in which there is a clear distinction between terminals and nonterminals. The second part of Section 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 λ RCB-grammars. For each class Φ of bounding functions we define Φ_m as the family of languages generated by λ RCB-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 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 Chomsky Normal Form for bounded λ RCB-grammars for one particular mode. The adjective “weak” means that in a weak CNF production $A \rightarrow XY$ not only $X, Y \in V - \Sigma$ is possible, but $X \in \Sigma$, or $Y \in \Sigma$ is also allowed.

In Section 4 we construct parsers for ϕ -bounded λ RCB/ 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.

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 recall some definitions and notational conventions from [8,9]. Then we introduce time-bounded RCB-grammars of which we give some examples. We conclude this section by establishing some properties of time-bounded RCB-grammars and their languages. For all unexplained notations and concepts from formal language theory we refer to standard texts like [1, 7, 11, 12].

Let $G = (V, \Sigma, P, S)$ be a context-free grammar with alphabet V , terminal alphabet Σ , set of productions P and initial symbol S . By $\bar{P}$ we denote the set of *reductions* corresponding to P , i.e., if an element π of P is equal to $A \rightarrow \alpha$, then $\bar{\pi}$ equals $\alpha \rightarrow A$ and $\bar{P} = \{\bar{\pi} | \pi \in P\}$. A member of $P \cup \bar{P}$ will be referred to as a *rule*.

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

- a context-free grammar $G = (V, \Sigma, P, S)$
- a regular language C with $C \subseteq (P \cup \bar{P})^*$.

G is called the *underlying grammar* of (G, C) and C is called the *control language* of (G, C) . The sentences of C will be called *control words*. $\square$

An RCB-grammar (G, C) ought to be provided with a mode of derivation denoted by m . We will now briefly discuss the several modes introduced in [8]. Each mode m results in a corresponding derivation relation $\Rightarrow_m$. We define the modes as follows. Every mode is determined by a list of three submodes separated by '/'. Each submode can vary over two values, which results in eight different modes. The first submode selects the nonterminal symbol from some string $\alpha \in V^*$ to which a production $\pi = A \rightarrow \sigma$ from P has to be applied.

- In the submode RN we select the right-most nonterminal symbol of α
- In the submode RO we select the right-most occurrence of the left-hand side of π in α .

If the selected nonterminal (determined by a certain submode) is equal to A then we say that π is applicable to α . We define a three-place predicate app_m on $P \times V^* \times V^*$ such that $app_m(\pi, \alpha, \beta)$ holds if and only if π is applicable to α with respect to the first submode of the mode m and β is the resulting string obtained from α by replacing this particular nonterminal A by the right-hand side σ of π . We extend this predicate to $(P \cup \bar{P}) \times V^* \times V^*$ by defining $app_m(\rho, \alpha, \beta) = app_m(\pi, \beta, \alpha)$, where ρ is a reduction with $\rho = \bar{\pi}$ for some unique $\pi \in P$.

Let c be a control word. We apply c to a string $\alpha \in V^*$ by successive application of the rules which constitute c . However, a rule r is not necessarily applicable to every intermediate string in a derivation. Now the second submode introduces two different ways to react to this phenomenon. Suppose that after some successful applications of rules prescribed by a prefix of c we have obtained a string β and that the next rule r of c is not applicable to β . First, we may choose to abort the application of c to α , and the resulting string will be undefined in that case. We call this the block mode (B-mode). In the skip mode (S-mode) we simply skip to the next rule of c , and then we try to apply this rule to the string β . For every rule r , the derivation relation $\Rightarrow_{m/B}^r$ over $V^* \times V^*$ is defined as

$$\alpha \Rightarrow_{m/B}^r \beta \text{ if and only if } app_{m/B}(r, \alpha, \beta)$$

and the derivation relation $\Rightarrow_{m/S}^r$ as

$$\begin{aligned} \alpha \Rightarrow_{m/S}^r \beta \text{ if and only if either } & app_{m/S}(r, \alpha, \beta) \\ & \text{or } \neg app_{m/S}(r, \alpha, \beta) \wedge (\alpha = \beta). \end{aligned}$$

The third submode concerns the way in which we deal with terminal productions used as reductions. In applying a reduction we will make a distinction between reductions ρ which either have (fair reductions) or have not

(general reductions) at least one nonterminal at their left-hand side. In fair mode (f-mode) we allow only fair reductions, and in general mode (g-mode) we allow the use of terminal productions as reductions as well. So each RCB-grammar ought to be provided with three different submode instances. For example ‘‘RCB/RN/S/f-grammar’’ or ‘‘RCB/RO/B/g-grammar’’ are correct ways to denote some types of RCB-grammars. Sometimes we do not specify one or more submodes, which will mean that both instances of the unspecified submodes are included. For example, ‘‘Q holds for the RN-mode’’ is the abbreviation for ‘‘Q holds for the RN/B/f, RN/B/g, RN/S/f and RN/S/g-mode’’.

For each control word c in $(P \cup \bar{P})^*$ we define the relation $\Rightarrow_m^c$ over $V^* \times V^*$ where m is a list of submodes, i.e., a mode. Viz. let $c = r_1 \dots r_n$, ($n \geq 0$, $r_i \in P \cup \bar{P}$, $1 \leq i \leq n$), then $\alpha \Rightarrow_m^c \beta$ holds if there exist strings $\alpha_i \in V^*$ ($1 \leq i \leq n-1$) with

$$\alpha \Rightarrow_m^{r_1} \alpha_1 \Rightarrow_m^{r_2} \alpha_2 \Rightarrow_m^{r_3} \dots \alpha_{n-1} \Rightarrow_m^{r_n} \beta.$$

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

We may omit the subscript m in $L_m(G, C)$ if the mode m is known from the context or if all modes are possible.

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 $\pi_1 = S \rightarrow AB$, $\pi_2 = A \rightarrow a$, $\pi_3 = B \rightarrow A$, $\pi_4 = A \rightarrow AA$, $\pi_5 = A \rightarrow b$. As the control language we take $C = \{c_1, c_2\}$ with $c_1 = \pi_1 \pi_2 \pi_3 \pi_4 \pi_5$ and $c_2 = \pi_1 \pi_2 \pi_3 \pi_2$. For each combination of submode instances mentioned above, in which the third submode equals g, 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. Now in c_1 $\bar{\pi}_4$ 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 $S \Rightarrow_{RO/S/g}^{c_1} ab$. $\square$

In [8] the following proposition concerning the generating power of the RCB-grammars has been proved.

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. $\square$

Example 2.5. 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.

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

The control language C is defined by

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

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, $\bar{\pi}_4 \pi_5 \pi_6$ 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 $(\pi_7 \bar{\pi}_8 \pi_9 \pi_6)^*$ 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 $\bar{\pi}_{10} \bar{\pi}_{11} \pi_{12} \pi_{10}$ 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. $\square$

We are now ready to introduce (time-)bounded RCB-grammars. We define the time function $T_{(G,C)}$ of an RCB-grammar (G, C) as 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 [6] for general phrase-structure grammars which has been investigated by Book [4]. 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 consider only RCB-grammars that are strictly λ -free (i.e., the underlying context-free grammar G has no λ -productions at all). As in [8] we refer to these grammars as λ RCB-grammars.

Definition 2.6. For any λ RCB-grammar (G, C) and every $w \in L(G, C)$, where $L(G, C) = \{w \in V^+ \mid \exists c \in C. S \Rightarrow^c w\}$, 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,

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

The function $t_{(G,C)}$ is partial recursive function. This is easy to show by modifying a similar proof from [4].

Definition 2.7. For every λ RCB-grammar (G, C) the *time function* $T_{(G,C)}: \mathbb{N} \rightarrow \mathbb{N}$ is the function determined by

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

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. [6]. In [4] 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 a underlying context-free grammar G . For some function $\phi: \mathbb{N} \rightarrow \mathbb{N}$, 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 $\forall n. T_{(G,C_1)}(n) \leq \phi(n)$ but not $\forall n. T_{(G,C_2)}(n) \leq \phi(n)$. The function ϕ will be called a *bounding function*.

Definition 2.8. 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 $F: A \rightarrow B$ we write $F(a) \downarrow$ whenever $F(a)$ is defined and $F(a) \uparrow$ otherwise.

Definition 2.9.

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

The family of ϕ -bounded λ RCB/ m -languages, denoted by $\mathbf{L}_m(\phi)$, consists of those languages for which there is a λ RCB/ 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 λ RCB/ m -languages — denoted by Φ_m — equals $\cup\{\mathbf{L}_m(\phi) \mid \phi \in \Phi\}$. $\square$

Example 2.10. The grammar (G, C) of Example 2.5 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 a length 2^m , for some $m \in \mathbb{N}$. Now we can make the following observations. $S \Rightarrow^{\pi_0} a$ and $S \Rightarrow^{\pi_1} aa$, so $1 \leq \phi(1)$, and $1 \leq \phi(2)$. We have $S \Rightarrow^{\pi_2 \pi_3^{m-2}} a^{m-1} Aaa$, $m \geq 2$, with $|\pi_2 \pi_3^{m-2}| = m-1$. Let Δ denote the set $\{\pi_4 \pi_5 \pi_6 (\pi_7 \pi_8 \pi_9 \pi_6)^* \pi_{10} \pi_{11} \pi_{12} \pi_{10}\}$ and let $d \in \Delta$. Then for $y \geq 2$ we have

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

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

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

such 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 $\psi: 1 \mapsto 1, 2 \mapsto 1, n \mapsto 4(n + 2 \log n - 4)$, where $n \geq 3$. $\square$

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

Lemma 2.11. *Let (G, C) be a λ RCB-grammar. Then for each mode m the following statements are equivalent.*

- (1) $L(G, C)$ is bounded by ϕ .
- (2) $\forall w \in L(G, C). \exists c \in C. (S \Rightarrow^c w \wedge |c| \leq \phi(|w|))$. $\square$

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

Lemma 2.12. *The family of λ RCB/RN/B/f-languages coincides with the family λ CFL.* $\square$

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

Proposition 2.13.

- (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 λ CFL $\subseteq$ LIN $_m \subseteq$ POLY(k) $_m \subseteq$ POLY $_m$.*
- (c) *For all modes $m \neq$ RN/B/f, we have λ 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 λ RCB/ 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: $A \rightarrow a$, $A \rightarrow aB$, $A \rightarrow aBC$, 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 [8] which generate L_0 . These grammars can easily be shown to be linearly bounded λ RCB/ m -grammars. For the mode RN/S/f Example 2.10 establishes the result. (In [8] also a λ RCB/RN/S/f-grammar has been constructed which generates L_0 . However, this grammar is bounded by a polynomial of degree two.) $\square$

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

Corollary 2.14. $\lambda\text{CFL} = \text{LIN}_{\text{RN/B/f}} = \text{POLY}(k)_{\text{RN/B/f}} = \text{POLY}_{\text{RN/B/f}}$.

Proof: This follows immediately from Proposition 2.13(a) and Lemma 2.12. $\square$

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 2.14 the family $\Phi_{\text{RN/B/f}}$ ($\Phi = \text{LIN}$, $\text{POLY}(k)$ 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 $\phi_i \in \text{POLY}(k)$ ($i = 1, 2$). 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 , $\text{POLY}(k)$ or POLY . Then the following statements hold.*

- *For all modes m , the families Φ_m are closed under union.*
- *The families $\Phi_{\text{B/f}}$ and the family $\Phi_{\text{RN/S/f}}$ are closed under marked concatenation and marked Kleene $+$.*
- *The families Φ_f are closed under marked concatenation.*
- *The families $\Phi_{\text{RO/f}}$ are closed under concatenation.*
- *The family $\Phi_{\text{RO/B/f}}$ 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 $\pi_i = S \rightarrow S_i$ ($i = 1, 2$). Define the regular control language C by $C = \{\pi_1\}C_1 \cup \{\pi_2\}C_2$. Then $L(G, C) = L(G_1, C_1) \cup L(G_2, C_2)$. 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 $\Phi = \text{POLY}_m(k)$ 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 $\lambda\text{RCB/B/f-}$ or $\lambda\text{RCB/RN/S/f-}$ 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 $L(G, C) = (L_1\#)^+$. 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

$$\begin{aligned} T_{(G,C)}(n) &\leq \sum_{i=1}^l (1 + \phi_1(s_i)) = \sum_{i=1}^l (1 + \sum_{j=0}^k a_j s_i^j) = l + \sum_{j=0}^k a_j \sum_{i=1}^l s_i^j \leq \\ &\leq l + \sum_{j=0}^k a_j (\sum_{i=1}^l s_i)^j + a_0(l-1) \leq \phi_1(n) + n(a_0 + 1), \end{aligned}$$

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 or $V_1 \cup V_2$. Finally, the control languages are concatenated (to the right) with $\bar{\Delta}\Omega$ and $\bar{\Delta}^*\Omega^*$, 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 from the case Φ equals $POLY(k)$ in a simple way. $\square$

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 [8] 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 $\lambda\text{RCB/RO}$ -grammars. Starting from a $\lambda\text{RCB/RO}$ -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 $\lambda\text{RCB/RO}$ -grammar (G, C) that generates $L(G_1, C_1) \cap R$. Here $G = (V, \Sigma, P, S)$ with $\Sigma = \Sigma_1 \cup \Sigma_R$ 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

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

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

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

where

$$P_0 = \{S \rightarrow Z(u, S_1, q_0) \mid u \in Q\},$$

$$P_F = \{A_a \rightarrow Z(u, a, t) \mid u = \delta(t, a), u \in F, a \in \Sigma_1\},$$

$$P_E = \{A_a \rightarrow a \mid a \in \Sigma\},$$

$$P_\Sigma = \cup \{P_a \mid a \in \Sigma\},$$

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

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

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

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

where for every p, q in Q

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

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

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 $\Phi_{\text{RO/B/f}}$ 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 λ RCB/RO/B/f-language and let σ be a λ RCB/RO/B/f-substitution $\sigma: \Sigma_1 \rightarrow 2^{\Sigma^*}$. Next, let $\Sigma_1 = \{a_1, \dots, a_n\}$ and for each $a \in \Sigma_1$, let (G_a, C_a) be a λ RCB/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 $i \neq j$ for every $1 \leq i, j \leq n$. Define alphabets $\Delta = \{S_{a_1}, \dots, S_{a_n}\}$ and $\Omega = \{Z_{a_1}, \dots, Z_{a_n}\}$. Let T be the control set $\cup \{C_a \mid a \in \Sigma_1\}$, and $U = \{A \rightarrow \alpha \mid A \in N_1, \alpha \in (N_1 \cup \Omega)^+\}$. We use the isomorphism $i: V_1 \rightarrow N_1 \cup \Omega$ defined by

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

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

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

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

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

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

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

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

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

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

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

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

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 λ RCB/RO-language and the substitution is a λ -free context-free substitution. The grammars (G_a, C_a) for $\sigma(a)$ are λ RCB/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 $\cup \{P_a | a \in \Sigma_1\} \cup h(P_1)$. We define U as $\{A \rightarrow \alpha | A \in N_1, \alpha \in (N_1 \cup \Delta)^*\}$ and the isomorphism i is defined by $i: V_1 \rightarrow N_1 \cup \Delta$ 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 $\Phi = \text{POLY}(k)$ as a corollary. $\square$

In [8] we introduced the following normal form.

Definition 3.4. 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: $A \rightarrow XY$ or $A \rightarrow a$ with $A \in N$ ($N = V - \Sigma$), whereas $X, Y \in V$ and $a \in \Sigma \cup \{\lambda\}$. An RCB-grammar or a λ RCB-grammar (G, C) is in *weak CNF* if its underlying grammar G is in weak CNF. $\square$

This definition can be adapted to time-bounded λ RCB-grammars in the obvious way. We showed that for every RCB/RN/B/f-grammar (G_0, C_0) there exists an equivalent RCB/RN/B/f-grammar in weak CNF [8,9]. The same holds for the time-bounded variant.

Proposition 3.5. Let Φ be a family of bounding functions. If Φ is equal to *LIN*, *POLY*(k) or *POLY*, then for every $\Phi_{\text{RN/B/f}}$ -grammar (G_0, C_0) there exists an equivalent $\Phi_{\text{RN/B/f}}$ -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. [8] 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 $P_1 = \{\pi_1, \dots, \pi_n\}$ be the set of productions of G_1 with $\pi_i = A_i \rightarrow B_{i,1} \dots B_{i,m_i}$. 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 $\pi_{i,1} = A_i \rightarrow B_{i,1} D_{i,1}$, $\pi_{i,2} = D_{i,1} \rightarrow B_{i,2} D_{i,2}$, $\dots$, $\pi_{i,m_i-1} = D_{i,m_i-2} \rightarrow B_{i,m_i-1} B_{i,m_i}$. 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 $h: P_1 \rightarrow P^*$ with $h(\pi_i) = \pi_i$ if $m_i \leq 2$ and $h(\pi_i) = \pi_{i,1}, \dots, \pi_{i,m_i}$ if $m_i \geq 3$. Furthermore, for a reduction $\bar{\pi} \in \bar{P}_1$ define $h(\bar{\pi}) = \bar{h}(\bar{\pi})$, using $\bar{\pi}\tau = \tau\bar{\pi}$ 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. $\square$

4. Parsing λ RCB-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 $\alpha \rightarrow A$ in the control language causes the parsing algorithm to rewrite the right-most nonterminal of the current sentential form of the parsing algorithm 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 $\underline{\alpha}$ if there is a string $\underline{\beta}$ such that $app_m(\bar{j}, \alpha, \beta)$ (assuming, of course, that $\pi = \pi$, 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 $\alpha \rightarrow A$ 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 sections 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 [12]. As in [12], we use a stack (here represented by $\mathbf{T}$) to handle the backtrack information.

Algorithm 4.1. *A depth-first bottom-up parser for λ RCB/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 $\Delta \subseteq P \cup \bar{P}$, that accepts C^R , i.e., the reverse of C .
 – string $w \in \Sigma^*$, where $w = w_1 \dots w_n$, $n \geq 1$, $w_i \in \Sigma$.
 – 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($[\lambda, w, 0, 0, \lambda, q_0], \mathbf{T}$)

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)  $j \in \text{Follow}(q)$ 
  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 $\text{rearrange}(u,v)$. 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 $\text{rearrange}(u,v)$. 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 $1 \dots 2 \cdot |P|$. 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 $\mathbf{T}$. 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 $PUSH([u, v, j, t, c, q], \mathbf{T})$. 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 $B \rightarrow xAy$ 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 changing 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 $\mathbf{M}$ 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 $\lambda RCB/RN/B/f$ -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 2.11 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 $\alpha A \rightarrow A$ in the control language, at most $\phi(n) - 1$ times, where $|\alpha| < \max\{|\gamma| \mid A \rightarrow \gamma \in P\}$. 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 $\mathbf{M}$ 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. $\square$

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 $\lambda\text{RCB}/m$ -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 $\lambda\text{RCB}/\text{RN}/\text{B}/\text{g}$ -languages can be obtained from Algorithm 4.1 by changing the part beginning at “ii) $j = xAy \rightarrow z$ with $u = pz$ and . . .” up to and including “ $u := pxAy$ ” into the following sequence of instructions.

```

ii)  $j = xAy \rightarrow z$  with  $u = pz$  and  $x, p \in V^*, z \in V^+, y \in \Sigma^*$ 
    or  $j = x \rightarrow A$  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
        (*  $j = x \rightarrow A, x \in \Sigma^+ *$ )
         $u := px$ 
    else
         $u := pxAy$ 
    end if

```

Concerning the time and space complexity, we can easily show that for the algorithm for $\lambda\text{RCB}/\text{RN}/\text{B}/\text{g}$ -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 $\lambda\text{RCB}/\text{RO}/\text{B}/\text{f}$ -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 $P \cup \bar{P}_f$, where

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

Each time a new state q' is computed from $\delta(j, q)$, *notapp* is set to the value $P \cup \bar{P}_f$, 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($[u, v, j, t, c, q, \text{Skip}, \text{notapp} - \{j\}], \mathbf{T}$).

Algorithm 4.3. *A depth-first bottom-up parser for λ RCB/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 $\Delta \subseteq P \cup \bar{P}$, that accepts C^R , i.e., the reverse of C .
 – string $w \in \Sigma^*$, where $w = w_1 \dots w_n$, $n \geq 1$, $w_i \in \Sigma$.
 – 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($[\lambda, w, 0, 0, \lambda, q_0, \text{false}, P \cup \bar{P}_f], \mathbf{T}$)
2. **repeat**
 $[u, v, i, t, c, q, \text{Skip}, \text{notapp}] := \text{POP}(\mathbf{T})$
 dead_end := false
 repeat
 if not *Skip* **then**
 Find the first rule j with $j > i$ that satisfies
 i) $j \in \text{Follow}(q)$
 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($[u, v, j, t, c, q, \text{Skip}, \text{notapp} - \{j\}], \mathbf{T}$)
 $u := pxAy$

```

    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 $P \cup \bar{P}_f$. Of course, no rule can be concatenated to the control string already found. The time counter t will not be increased too. 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 Algorithm 4.1; cf. the proof of Proposition 4.2.

As a last example of λ RCB/ 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

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

into

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

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

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

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 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 paper we applied the idea of time-bounded grammars, as introduced in [4, 6], to the concept of λ RCB-grammar [8]. 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 $\lambda CFL = \Phi_{RN/B/f}$, 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 bounding functions. Let $\mathbf{C}$ 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 $\mathbf{C}$ and of Φ which imply a certain closure property of the family of languages generated by Φ -bounded $\mathbf{C}$ -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.

The meaning of the assumptions on the family of bounding functions Φ mentioned in Table 3 are listed in Table 2. To obtain closure properties for the family of Φ -bounded λ -free $\mathbf{C}$ -controlled bidirectional languages we often need closure under left- or right-marking. A family of languages $\mathbf{C}$ is closed under *left-* and *right-marking* if for every language $L_0 \in \mathbf{C}$ also $\{\#\}L_0 \in \mathbf{C}$ and $L_0\{\#\} \in \mathbf{C}$, respectively, where $\#$ does not occur in the alphabet of L_0 . 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. Because $\mathbf{C}$ is no longer equal to

μ	$\forall p \geq 2. \forall \psi_1, \dots, \psi_p \in \Phi. \exists \phi \in \Phi. \forall n. \phi(n) \geq \max\{\psi_i(n) \mid 1 \leq i \leq p\}$
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 real numbers.

Table 2.

Assumptions on $\mathbf{C}$	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 family of regular languages, we generalize λ RCB-grammars to λ -free $\mathbf{C}$ -controlled bidirectional grammars (λ CCB-grammars). Then $\Phi_m(\mathbf{C})$ denotes the family of languages generated by λ CCB/ m -grammars that are bounded by some bounding function from Φ . Since most of the closure properties of the family of $\Phi_{RN/B/f}$ -languages heavily depend on $\mathbf{C}$ being the family of regular control languages – cf. Proposition 2.4.(2) in [8] – we cannot expect to maintain all the closure properties if we generalize to a more general family $\mathbf{C}$ of control languages.

Acknowledgment. I wish to thank Peter Asveld for his aid during the preparation of this paper.

References

1. A.V. Aho & J.D. Ullman: *The Theory of Parsing, Translations and Compiling – Volume I: Parsing* (1972), Prentice-Hall, Englewood Cliffs, N.J.
2. 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.
3. L. Boasson & G. Sénizergues: NTS languages are deterministic and congruential, *J. Comput. System Sci.* **31** (1985) 332-342.
4. R.V. Book: Time-bounded grammars and their languages, *J. Comput. System Sci.* **5** (1971) 397-429.
5. R.V. Book: Thue systems as rewriting systems, *J. Symb. Comp.* **3** (1987) 39-68.
6. 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).
7. M.A. Harrison: *Introduction to Formal Language Theory* (1978), Addison-Wesley, Reading, Mass.
8. J.A. Hogendorp: Controlled bidirectional grammars, Memorandum INF-88-16, Department of Computing Sciences, University of Twente. To appear in *Internat. J. Comput. Math.* **28** nr.1.
9. J.A. Hogendorp: Controlled rewriting using productions and reductions, Proceedings of “Computer Science in the Netherlands – 1988” (1988) 479-494.
10. R. McNaughton, P. Narendran & F. Otto: Church-Rosser Thue systems and formal languages, *J. Assoc. Comput. Mach.* **35** (1988) 324-344.
11. A. Salomaa: *Formal Languages* (1973), Academic Press, New York.
12. T.A. Sudkamp: *Languages and Machines — An Introduction to the Theory of Computer Science* (1988), Addison-Wesley, Reading, Mass.