

On the Generating Power of Regularly Controlled Bidirectional Grammars

Peter R.J. Asveld & Jan Anne Hogendorp*

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

Abstract — RCB-grammars or regularly controlled bidirectional grammars are context-free grammars of which the rules can be used in a productive and in a reductive fashion. In addition, the application of these rules is controlled by a regular language. Several modes of derivation can be distinguished for this kind of grammar. In this paper the generating power of the derivation mode that uses right-occurrence rewriting (RO-mode) is determined. Furthermore, a new mode called RA is introduced, which is a better formalization of the intuitive idea of right-occurrence rewriting than the RO-mode. The RO- and RA-mode have the same generating power, viz. the corresponding RCB-grammars both generate the recursively enumerable languages. Consequently, providing RCB/RO-grammars with a time bound results in a less powerful grammar model.

1. Introduction

Context-free grammars in which the rules can be used in a productive as well as in a reductive fashion have been studied by several authors. Cf. [3, 4, 7, 18], in which the attention was focused on the so-called NTS or nonterminal separating property: a context-free grammar has the NTS-property if each sentential form that can be derived from a nonterminal by means of both productions and reductions can also be derived by the use of productions only.

In [12] the NTS-property inspired the introduction of RCB or regularly controlled bidirectional grammars. In an RCB-grammar the productions can be used in both directions, i.e., as productions and as reductions. However, the application of the rules is controlled by a regular language C of control words, which consist of productions and reductions. Formally, an RCB-grammar is denoted as a pair (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 corresponding to P ; viz. $\bar{P} = \{\alpha \rightarrow A \mid A \rightarrow \alpha \in P\}$. If π denotes a production in P , then $\bar{\pi}$ denotes the corresponding reduction in \bar{P} . In the sequel, an element of $P \cup \bar{P}$ will be called a rule. For this kind of grammar we distinguish several modes of derivation – cf. [12, 13] – which are described in Section 2. By varying three different aspects of the derivational process, each over two instances, we obtain eight different modes of derivation. In this paper the generating power of RCB-grammars combined with these

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

modes of derivation are studied. For each mode m an RCB-grammar (G, C) gives rise to a language $L_m(G, C)$. Therefore we obtain for each mode m a corresponding language family.

One of these aspects of the derivational process is the selection of the terminal that has to be rewritten – if possible – by the next rule prescribed by the control word. In [12] the right-occurrence or RO-mode has been introduced: in RO-mode a production π of the control word has to be applied to the right-most occurrence in a string α of the left-hand side of π . In addition we can apply a reduction $\bar{\pi}$ in RO-mode to a string α , obtaining a string β if and only if we can apply π in RO-mode to β , and the result of this application is equal to α . In [12] we introduced this rather “exotic” way of rewriting in order to establish some closure properties of RCB-languages, viz. closure under homomorphism, inverse homomorphism, intersection with a regular set, and under context-free substitution. Now the main result of the present paper 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.

The paper is organized in the following way. In Section 2 we recall the concept of RCB-grammar, and the different modes of derivation. We also repeat the definition of Turing machine 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 [14] are weaker than ordinary RCB/RO-grammars with respect to generating power. This follows from the fact that time-bounded RCB-languages are recursive [14]. Then in Section 5 we discuss a few new modes of derivation, of which the so-called 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 [11, 15] for all unexplained notations and concepts from formal languages and complexity theory. Another useful standard text is [16]. First, we recall some definitions and terminology from [12].

As usual $G = (V, \Sigma, P, S)$ denotes a context-free grammar with alphabet V , terminal alphabet Σ , set of productions P and initial symbol S . The set \bar{P} consists of the *reductions* corresponding to P , i.e., for every production π in P with $\pi = A \rightarrow \alpha$ we have $\bar{\pi}$ in \bar{P} , with $\bar{\pi} = \alpha \rightarrow A$. Hence $\bar{P} = \{\bar{\pi} | \pi \in P\}$. A member of $P \cup \bar{P}$ will be called a *rule*. The empty word will be denoted by λ .

Definition 2.1. A *regularly controlled bidirectional grammar* or *RCB-grammar* is a pair (G, C) where

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

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

An RCB-grammar (G, C) will be provided with a mode of derivation denoted by m [12]. Each mode m results in a corresponding derivation relation \Rightarrow_m . Every mode is determined by a list of three submodes separated by “/”. Each submode can vary over two instances, which results in eight different modes.

The first submode prescribes which nonterminal symbol of a sentential form has to be rewritten. We distinguish the following selection mechanisms. Let α ($\alpha \in V^*$) be a sentential form and π a production from P , with $\pi = A \rightarrow \sigma$.

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

For each submode, RN or RO, we say that the production π is *applicable* to α if the selected nonterminal is equal to the left-hand side of π . In addition, we say that a reduction $\bar{\pi}$ is *applicable* to α if and only if there is a string β such that π is applicable to β and the result of the application of π to the selected nonterminal is α .

A control word c in C consists of a sequence of rules from $P \cup \bar{P}$. The application of c to a string α in V^* is performed by successive application of the rules which constitute c . The case that a rule in c is not applicable leads to the second submode with the following two instances. Either we skip the rule and try to apply the next rule of c to α (*skip mode* or *S-mode*), or we block further application of any rule from c (*block mode* or *B-mode*). In the latter case the application of c to α gives no result, i.e., there is no string β such that $\alpha \Rightarrow_{m/B}^c \beta$ is defined, where m is equal to RO or RN. In the S-mode we have that if a rule r is not applicable, then $\alpha \Rightarrow_{m/S}^r \alpha$ holds (m is again equal to RO or RN).

Finally, the third submode concerns the distinction between reductions that either have or have not a terminal production as its base. A reduction based on a terminal production is called a *terminal reduction*. And a reduction based on a production that is not terminal – i.e., the right-hand side possesses at least one nonterminal symbol – is called a *fair reduction*. In the *general mode* (*g-mode*) we allow both kinds of reduction, and in the *fair mode* (*f-mode*) we only allow fair reductions.

So each RCB-grammar will be provided with three different submode instances. For example, an RCB-grammar with right-occurrence derivation, block mode and allowing general reductions is a RCB/RO/B/g-grammar. It is possible to leave some or even all submode instances unspecified. This is intended as a shorthand for long phrases. 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 on 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

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

We can now define the language generated by an RCB-grammar for each of the concrete modes of derivation defined above.

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

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

and \mathbf{L}_m denotes the family of languages generated by RCB/ m -grammars. \square

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

Example 2.3. Consider the following RCB-grammar (G, C) with $G = (\{S, A, B, a, b, d\}, \{a, b, d\}, P, S)$ and P consists of

$$\begin{array}{llll} \pi_1 = S \rightarrow AB & \pi_3 = B \rightarrow A & \pi_5 = A \rightarrow b & \pi_7 = S \rightarrow d \\ \pi_2 = A \rightarrow a & \pi_4 = A \rightarrow AA & \pi_6 = B \rightarrow Bb & \pi_8 = S \rightarrow SAB. \end{array}$$

Define control words c_1, c_2, c_3 by $c_1 = \pi_1 \pi_3 \pi_4 \pi_5$, $c_2 = \pi_1 \pi_2 \pi_3 \pi_5$ and $c_3 = \pi_1 \pi_2 \pi_3 \pi_4 \pi_4 \pi_2 \pi_2$, and let C_0 denote the set $\{(\pi_8 \pi_6 \pi_3 \pi_4 \pi_5)^+ \pi_7\}$. As control language we take $C = C_0 \cup \{c_1, c_2, c_3\}$. From Table 1 it follows that – apart from the third submode – for each combination of submode instances mentioned above we obtain a different language. Due to the control language C the value of the third submode happens to be immaterial for $L_m(G, C)$. \square

In [12] the following result concerning the generating power of 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

For some examples of RCB-languages that are not context-free we refer to [12, 13]. For instance, the languages $\{a^{2^n} \mid n \geq 0\}$ and $\{a^n b^n c^n \mid n \geq 1\}$ are non-context-free RCB/RN/S/f-languages.

Now we recall a definition of Turing machine and some related concepts from [11].

Definition 2.5. A *deterministic single-tape Turing machine* is a 7-tuple $A = (Q, \Sigma, \Gamma, B, \delta, q_0, F)$, where

$S \Rightarrow_m^c w$			$L_m(G, C)$
m	w	c	
RN/B	b —	c_1 otherwise	$\{b\}$
RN/S	b Ab aa db	c_1 c_2 c_3 $c \in C_0$	$\{a^2, b, db\}$
RO/B	b ab —	c_1 c_2 otherwise	$\{ab, b\}$
RO/S	b ab aaa db	c_1 c_2 c_3 $c \in C_0$	$\{a^3, ab, b, db\}$

Table 1.

- 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*. \square

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 2.6. 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 $\Gamma^* F \Gamma^+$. \square

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 2.7. 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 $x = \alpha a q b \beta$ 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
- $(d = +1 \text{ and } \alpha' = \alpha ac \text{ and } \beta' = \beta)$ or
 $(d = -1 \text{ and } \alpha' = \alpha \text{ and } \beta' = ac \beta)$ or
 $(d = 0 \text{ and } \alpha' = \alpha a \text{ and } \beta' = c \beta)$.

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

Definition 2.8. 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 [11, 15, 16] 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 [11] 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.

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

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

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

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 $\{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6\}$, $P_{\Sigma\Sigma}$, $P_{\$R}$ and $P_{\$L}$ of P consist of productions that initialize the simulation of the Turing machine A . These productions are defined by

$$\begin{aligned} \pi_1 &= S \rightarrow S(\lambda, B), & \pi_2 &= S \rightarrow \$q_0(\lambda, B), \\ \pi_3 &= S \rightarrow W_\$U, & \pi_4 &= U \rightarrow \$, \\ \pi_5 &= W \rightarrow W_\$ \$, & \pi_6 &= W \rightarrow \$q_0. \end{aligned}$$

Furthermore,

$$\begin{aligned} P_{\Sigma\Sigma} &= \{U \rightarrow (a, a)U \mid a \in \Sigma\}, \\ P_{\$R} &= \{W_a \rightarrow (a, a)\$ \mid a \in \Sigma\}, \\ P_{\$L} &= \{W_a \rightarrow \$(a, a) \mid a \in \Sigma\}. \end{aligned}$$

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 $\bar{P}_{i,I}$ will be used. Then the rules in the corresponding set P_i will actually complete that simulation.

$$\begin{aligned} P_{0,I} &= \{(p, a, D) \rightarrow p(a, D) \mid a \in \Sigma \cup \{\lambda\}, p \in Q, D \in \Gamma, \\ &\quad \exists E \in \Gamma, \exists q \in Q \cdot \delta(p, D) = (q, E, 0)\}, \\ P_0 &= \{(p, a, D) \rightarrow q(a, E) \mid a \in \Sigma \cup \{\lambda\}, p, q \in Q, D, E \in \Gamma, \delta(p, D) = (q, E, 0)\}, \\ P_{1,I} &= \{(p, a, D) \rightarrow p(a, D) \mid a \in \Sigma \cup \{\lambda\}, p \in Q, D \in \Gamma, \\ &\quad \exists E \in \Gamma, \exists q \in Q \cdot \delta(p, D) = (q, E, 1)\}, \\ P_1 &= \{(p, a, D) \rightarrow (a, E)q \mid a \in \Sigma \cup \{\lambda\}, p, q \in Q, D, E \in \Gamma, \delta(p, D) = (q, E, 1)\}, \\ P_{-1,I} &= \{(b, H, p, a, D) \rightarrow (b, H)p(a, D) \mid a, b \in \Sigma \cup \{\lambda\}, p \in Q, D, H \in \Gamma, \\ &\quad \exists E \in \Gamma, \exists q \in Q \cdot \delta(p, D) = (q, E, -1)\}, \\ P_{-1} &= \{(b, H, p, a, D) \rightarrow q(b, H)(a, E) \mid a, b \in \Sigma \cup \{\lambda\}, p, q \in Q, \\ &\quad D, E, H \in \Gamma, \delta(p, D) = (q, E, -1)\}. \end{aligned}$$

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.

$$P_R = \{(q, a, D) \rightarrow q(a, D) \mid q \in F, a \in \Sigma \cup \{\lambda\}, D \in \Gamma\},$$

$$P_L = \{(q, a, D) \rightarrow (a, D)q \mid q \in F, a \in \Sigma \cup \{\lambda\}, D \in \Gamma\},$$

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

$$P_\lambda = \{q \rightarrow \lambda \mid q \in F\}.$$

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 $S \Rightarrow_{RO/S/f}^c w$, then the string c' obtained from c by removing each skipped rule has the property $S \Rightarrow_{RO/B/f}^{c'} w$ and $c' \in C$.

The construction sketched above works as follows. If the Turing machine A accepts the string $a_1 \dots a_n$, 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 $\pi_1 = S \rightarrow S(\lambda, B)$ to S , followed by either π_2 or π_3 in order to remove S . In the latter case this production is followed by zero or more applications of productions of the form $U \rightarrow (a, a)U$ with $a \in \Sigma$, and a single application of the production $U \rightarrow \$$. Thus there exists a control string c_1 in $\{\pi_1\}^* (\{\pi_2\} \cup \{\pi_3\} P_{\Sigma\Sigma}^*) \{\pi_4\}$ such that

$$S \Rightarrow_{RO/f}^{c_1} W_\$(a_1, a_1) \dots (a_n, a_n) \$ (\lambda, B)^k, \quad (n+k \geq 1).$$

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

By zero or more applications of pairs of the form $(a, a)\$ \rightarrow W_a$ and $W_a \rightarrow \$(a, a)$ with $a \in \Sigma$, and followed by the application of π_5 and π_6 we observe that there exists a control string c_2 in $(\bar{P}_{\$R} P_{\$L})^* \{\pi_5 \pi_6\}$ such that

$$\alpha_{n,k} \Rightarrow_{RO/f}^{c_2} \$q_0(a_1, a_1) \dots (a_n, a_n) (\lambda, B)^k, \quad (n+k \geq 1). \quad (1)$$

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

Note that inserting productions and reductions from $P_{\Sigma\Sigma} \cup P_{\$R} \cup P_{\$L} \{\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6\}$ 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. $\bar{P}_{0,I} P_0$ simulates an action of A with no head movement, $\bar{P}_{1,I} P_1$ takes care of a movement to the right A , and finally $\bar{P}_{-1,I} P_{-1}$ 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 $\bar{P}_{0,I}$, $\bar{P}_{1,I}$ and $\bar{P}_{-1,I}$ 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 $c_i \in \bar{P}_{i,I} P_i$ ($i = -1, 0, 1$) such that

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

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

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

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

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

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 $\bar{P}_{0,I}P_0$ such that

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

and

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

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 $\bar{P}_{1,I}P_1$ and $\bar{P}_{-1,I}P_{-1}$. 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 $(\cup\{\bar{P}_{i,I}P_i \mid i = -1, 0, 1\})^*$ we have

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

where $a_i = \lambda$ ($i > n$) and $X_i \in \Gamma$ ($1 \leq i \leq n+k$). Let the derived string in (2) be denoted by $\tilde{X}_{r,q}^{n+k}$.

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

$$\tilde{X}_{r,q}^{n+k} \Rightarrow_{RO/f}^d \$ a_1 \dots a_n q. \quad (3)$$

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

Thus $\{\$\}T(A) \subseteq L_{RO/f}(G,C)$. The converse inclusion can be proved by induction in a similar way.

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 $C = (P \cup \bar{P}_f)^*$, where \bar{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 $h: \Sigma \cup \{\$\} \rightarrow \Sigma^*$ 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 [13]), 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. \square

In the construction applied in the proof of Proposition 3.1 we defined the control language C equal to $(P \cup \bar{P})^*$ 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 $C = (P \cup \bar{P})^*$.

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

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 [1, 2, 3, 6, 8, 9, 17] are a few instances only.

4. Time-Bounded λ -free RCB-grammars

Originally, time-bounds have been introduced in [10] to describe the derivational complexity of general phrase-structure grammars. But they also provide a suitable method to associate a language family with a family of grammars and a class of functions; cf. [5] and [14] which are based on type-0 grammars and on λ -free RCB-grammars, respectively. A *λ -free RCB-* or *λ RCB-grammar* is an RCB-grammar that has no λ -productions. We recall some concepts from [14], which are slight modifications of basic notions introduced in [5, 10]. Let $|c|$ denote the length of the string c .

Definition 4.1. Let (G, C) be a λ RCB-grammar with $G = (V, \Sigma, P, S)$, and let $\underline{L}(G, C) = \{w \in V^+ \mid \exists c \in C. S \Rightarrow^c w\}$. Then

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

and the *time function* of (G, C) – denoted by $T_{(G,C)}: \mathbb{N} \rightarrow \mathbb{N} -$ is

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

A function $\phi: \mathbb{N} \rightarrow \mathbb{N}$ is called a *bounding function* if ϕ is a nondecreasing total recursive function such that there is a k ($k \in \mathbb{N}$) with for all x , $\phi(x) \geq x/k$ and for all $x \geq 0$, $\phi(x) \geq 0$.

A λ RCB-grammar (G, C) is *bounded* by a function ϕ if for each natural number n , $T_{(G,C)}(n) \leq \phi(n)$, provided that $T_{(G,C)}(n)$ is defined. And a λ RCB-language L_0 is *bounded* by a function ϕ if there exists a λ RCB-grammar (G, C) , generating L_0 , that is bounded by ϕ .

The family of ϕ -bounded λ RCB/ m -languages – denoted by $\mathbf{L}_m(\phi)$ – consists of all languages for which there exists 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

Let Φ denote a family of bounding functions. As in [14] – to which we also refer for some examples of languages generated by time-bounded λ RCB-grammars – we will mainly consider 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*.

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 4.2.

- (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 [14] parsing algorithms for Φ_{RO} -languages have been outlined. Since these algorithms terminate for each input, it follows that Φ_{RO} -languages are recursive. \square

Actually, we can slightly improve upon Corollary 4.2, for which we need the following concepts and notations. Let $\mathbf{NTIME}(\phi)$ be the family of λ -free languages which are accepted by multi-tape nondeterministic Turing machines in time $\phi: \mathbb{N} \rightarrow \mathbb{N}$. As usual \mathbf{NP} is defined by

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

i.e., \mathbf{NP} is the family of λ -free languages acceptable nondeterministically in polynomial time.

Proposition 4.3. (1) Let L_0 be a λ RCB-language bounded by a function $\phi: \mathbb{N} \rightarrow \mathbb{N}$. Then $L_0 \in \mathbf{NTIME}(\phi^2)$.

(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 $c \cdot \phi(n)$ 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. \square

It remains an open question whether a kind of converse of Proposition 4.3(2) holds, i.e., whether there exists a mode m such that $\mathbf{NP} \subseteq \mathbf{POLY}_m$. 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 $C = (P \cup \bar{P})^*$ by

$$\{\pi_1\}^* (\{\pi_2\} \cup \{\pi_3\} P_{\Sigma}^*) \{\pi_4\} (\bar{P}_{\$R} P_{\$L})^* \{\pi_5 \pi_6\} (\bar{P}_{-1,I} P_{-1} \cup \bar{P}_{0,I} P_0 \cup \bar{P}_{1,I} P_1)^* \\ (P_R \cup P_L \cup \bar{P}_R \cup \bar{P}_L \cup P_{\Sigma})^* P_{\lambda}.$$

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 $\mathbf{NTIME}(n) \subseteq \mathbf{LIN}$.

5. New Modes of Derivation.

In this section we introduce a new mode of derivation, the RA-mode. When we apply a production under RO-mode to a sentential form, 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 are possibly more than one substring that can be rewritten. For example, in the string aBa the reduction $a \rightarrow A$ is applicable to both a 's, i.e., we have $aBa \Rightarrow_{RO}^{a \rightarrow A} aBA$ and $aBa \Rightarrow_{RO}^{a \rightarrow A} ABA$. We want to model a mode of derivation in which both productions and reductions can rewrite at most one substring of a sentential form, such that the intuitive idea of right-occurrence rewriting is preserved.

Definition 5.1. 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. \square

In the example presented above, only the a on the right can be rewritten, i.e., $aBa \Rightarrow_{RA}^{a \rightarrow A} aBA$. There is another difference with the RO-mode, viz., $aBa \Rightarrow_{RA}^{a \rightarrow A} ABA$ 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 5.2. *A language L_0 is an RCB/RA-language if and only if L_0 is recursively enumerable; i.e., $\mathbf{L}_{RA} = \mathbf{RE}$.*

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. \square

In case of ordinary context-free grammars the RO-mode and RA-mode are also equivalent of course. Definition 5.1 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. Let Ξ be an alphabet. A *Thue system* T on Ξ is a finite subset of $\Xi^* \times \Xi^*$ and each element (u, v) of T is a rewriting rule. Now consider P as a Thue system with alphabet V and the relation \leftrightarrow_P defined by: for $u \rightarrow v \in P$ and $x, y \in V^*$, $xuy \leftrightarrow_P xvy$ and $xvy \leftrightarrow_P xuy$. Then we have

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

- $xuy \leftrightarrow_P xvy$
- u occurs in uy only once.

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

$xuy \Rightarrow_{RO}^u \overset{v}{\rightarrow} xvy$ 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 adopt this modification of RO into RA to the RN-mode as well. Let $\Sigma \subset V$. The RN-mode can be prescribed by

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

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

This allows us to write $Baa \Rightarrow_{RN/g}^a \overset{A}{\rightarrow} BAa$, as well as $Baa \Rightarrow_{RN/g}^a \overset{A}{\rightarrow} BaA$, where one would expect only the latter possibility. To obtain this effect, we define a new mode called *right-most string* or *RS-mode*.

Definition 5.3. Let α be a string in V^* and r a rule in $P \cup \bar{P}$. In the *RS-mode* the right-most occurrence of the left-hand side of r is selected as the string that ought to be rewritten, provided that the string to the right of this occurrence is in Σ^* . \square

In the terminology of Thue systems the RS-mode mode is characterized by

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

- $xuy \leftrightarrow_P xvy$
- u occurs in uy only once
- $y \in \Sigma^*$.

It is obvious that the following holds.

Proposition 5.4. *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. \square

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. We observe that the properties of the families of RN/B/g- and RN/S/g-languages, established in [11], also hold for the corresponding families of RS-languages.

Proposition 5.5. *The families of RS/B/g- and RS/S/g-languages are closed under union and in particular under union with a regular set.* \square

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 2 in which CF 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 CF (Proposition 2.4) and are, of course, included in RE.

m	RO				RN			
	B		S		B		S	
	f	g	f	g	f	g	f	g
L_m	RE	RE	RE	RE	CF	?	?	?

Table 2.

Whether there exist a 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.

Acknowledgment. We are indebted to Rieks op den Akker for some discussions concerning modes of derivation.

References

1. P.R.J. Asveld: Complete symmetry in D2L systems and cellular automata, *Internat. J. Comput. Math.* **19** (1986) 211-223.
2. B.S. Baker & R.V. Book: Reversal-bounded multipushdown machines, *J. Comput. System Sci.* **8** (1974) 315-332.
3. 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.
4. L. Boasson & G. Sénizergues: NTS languages are deterministic and congruential, *J. Comput. System Sci.* **31** (1985) 332-342.
5. R.V. Book: Time-bounded grammars and their languages, *J. Comput. System Sci.* **5** (1971) 397-429.
6. R.V. Book: Simple representation of certain classes of languages, *J. Assoc. Comput. Mach.* **25** (1978) 23-31.
7. R.V. Book: NTS grammars and Church-Rosser systems, *Inform. Process. Lett.* **13** (1981) 73-76.
8. K. Culik II: A purely homomorphic characterization of recursively enumerable sets, *J. Assoc. Comput. Mach.* **26** (1979) 345-350.
9. J. Engelfriet & G. Rozenberg: Fixed point languages, equality languages, and representation of recursively enumerable languages, *J. Assoc. Comput. Mach.* **27** (1980) 499-518.
10. A.V. Gladkii: On the complexity of derivations in phrase-structure grammar, *Algebr i Logika Sem.* **3** (1964) 29-44 (in Russian).
11. M.A. Harrison: *Introduction to Formal Language Theory* (1978), Addison-Wesley, Reading, Mass.
12. J.A. Hogendorp: Controlled bidirectional grammars, *Internat. J. Comput. Math.* **27** (1989) 159-180.
13. J.A. Hogendorp: Controlled rewriting using productions and reductions, *Proceedings of Computer Science in the Netherlands – 1988*, Utrecht (1988) 479-494.
14. J.A. Hogendorp: Time-bounded controlled bidirectional grammars, Memorandum INF-88-60 (1988), University of Twente, Enschede, The Netherlands.
15. J.E. Hopcroft & J.D. Ullman: *Introduction to Automata Theory, Languages, and Computation* (1979), Addison-Wesley, Reading, Mass.
16. A. Salomaa: *Formal Languages* (1973), Academic Press, New York.
17. W.J. Savitch: How to make arbitrary grammars look like context-free grammars, *SIAM J. Comput.* **2** (1973) 174-182.

18. G. Sénizergues: The equivalence and inclusion problems for NTS languages, *J. Comput. System Sci.* **31** (1985) 303-331.