Nonterminal complexity of programmed grammars

Henning Fernau

WSI-2000-26

Henning Fernau
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen
Sand 13
D-72076 Tübingen
Germany

email:fernau@informatik.uni-tuebingen.de
Telefon: (07071) 29-77565
Telefax: (07071) 29-5061

ISSN 0946-3852

Nonterminal complexity of programmed grammars

Henning Fernau
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen
Sand 13
D-72076 Tübingen
Germany
email: fernau@informatik.uni-tuebingen.de

December 30th, 2000

Abstract

We show that, in the case of context-free programmed grammars with appearance checking working under free derivations, three nonterminals are enough to generate every recursively enumerable language. This improves the previously published bound of eight for the nonterminal complexity of these grammars. This also yields an improved nonterminal complexity bound of four for context-free matrix grammars with appearance checking. Moreover, we establish an upperbound of four on the nonterminal complexity of context-free programmed grammars without appearance checking working under leftmost derivations of type 2. We derive nonterminal complexity bounds for context-free programmed and matrix grammars with appearance checking or with unconditional transfer working under leftmost derivations of types 2 and 3, as well. More specifically, a first nonterminal complexity bound for context-free programmed grammars with unconditional transfer (working under leftmost derivations of type 3) which depends on the size of the terminal alphabet is proved.

1 Introduction

Descriptional complexity (or, more specifically, syntactic complexity) is interested in measuring the complexity of describing objects (in our case, formal languages) with respect to different syntactic complexity measures. In particular, very economical presentations of languages are sought for. For example, Shannon [21] showed the nowadays classical result that every recursively enumerable language can be accepted by some Turing machine with only two states.

Similar complexity considerations may be carried out for any language describing device. In the case of grammars, natural syntactic complexity measures are the number of nonterminals and the number of rewriting rules. In this paper, we will consider the nonterminal complexity of certain regulated grammar formalisms which characterize the recursively enumerable languages. In the literature, several interesting results on this topic appeared in recent years. For example, in the case of scattered context grammars, there has even been some sort of race for the smallest possible complexity bound, see [14, 15, 16]. Here , we will concentrate on the question: how many nonterminals must a context-free programmed grammar (working under free derivation) with appearance checking necessarily have in order to be able to generate every recursively enumerable language? Previously, a solution using eight nonterminals has been known [4, Theorem 4.2.3]. We improve this bound to three by using a rather intricate Turing machine simulation. This is our main result. This result could also be useful within the emerging area of membrane computing [19]. As a corollary, we derive that three nonterminals are enough to generate every recursively enumerable language by using context-free programmed grammars with appearance checking working under leftmost derivations of type 3. The same bound was previously claimed for context-free programmed grammars without appearance checking working under leftmost derivations of type 2 by Meduna and Horváth [17, Theorem 5] (within Kasai's formalism of state grammars [13]). Since we think that the proof given there is incorrect, we give a new characterization of the recursively enumerable languages through programmed grammars without appearance checking working under leftmost derivations of type 2 with four nonterminals based on the construction leading to our main theorem. Similarly, a nonterminal bound of four can be derived for grammars with unconditional transfer checking working under leftmost derivations of type 2.

Our main result also yields an improved nonterminal complexity bound for context-free matrix grammars with appearance checking (namely four instead of six as previously published in [18], also see [4, Theorem 4.2.3]; independently, this bound was achieved recently by Freund and Păun [9]). This bound holds for matrix grammars working under free derivations and working under leftmost derivations of type 3, as well.

Finally, we can derive a first nonterminal complexity bound for context-free programmed (or matrix) grammars with unconditional transfer (working under leftmost derivations of type 3), under the assumption that the terminal alphabet is fixed.

We use standard mathematical and formal language notations throughout the paper, as they can be found in [4, 12]. In particular: π_j selects the jth component of an n-tuple; λ denotes the empty word; w^R denotes the reversal of string w. Moreover, we will *not* consider two languages L_1 and L_2 equal if $L_1 \setminus \{\lambda\} = L_2 \setminus \{\lambda\}$.

The paper is organized as follows. Section 2 introduces Turing machine as generators of formal languages, a formalism rarely encountered in the literature, albeit it is natural and adequate for our purposes. In Section 3, we introduce the notion of programmed grammars which is basic for the whole paper. Section 4

contains the proof of the main result of this paper, namely, that three nonterminals are enough to generate all recursively enumerable languages by means of context-free programmed grammars with appearance checking. In Section 5, we explain the consequences of our main result for the nonterminal complexity of programmed grammars with leftmost derivations of types 3 and 2 and of matrix grammars. Section 6 discusses regulated grammars with unconditional transfer. Finally, we consider the question wether our main result could be further strengthened.

2 Turing machines

In order to be able to reason more formally, in the following, we give a definition of a Turing machine which is adapted to our purposes. The reader can probably easily check its equivalence with his or her favourite definition.

Definition 1 A *(nondeterministic) Turing machine (with one one-sided tape)* is given by

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_f, \#_L, \#_R, \#),$$

where Q is the state alphabet, Σ is the input alphabet, Γ (with $\Sigma \subseteq \Gamma$ and $\Gamma \cap Q = \emptyset$) is the tape alphabet, $\delta \subseteq Q \times \Gamma \times \{L, R\} \times Q \times \Gamma$ is the transition relation, q_0 is the initial state, q_f is the final state, $\#_L \in \Gamma$ is the left endmarker, $\#_R \in \Gamma$ is the right endmarker and $\# \in \Gamma$ is the blank symbol.

A *configuration* (also called instantaneous description) of M is described by a word $c \in \#_L\tilde{\Gamma}^*\#_R Q \cup \#_L\tilde{\Gamma}^* Q \tilde{\Gamma}^*\#_R$ with $\tilde{\Gamma} = \Gamma \setminus \{\#_L, \#_R\}$. Here $c = \#_L wqv$ means: The head of the Turing machine is currently scanning the last symbol a of $\#_L w$. We now describe possible (and the only possible) successor configurations c' of $c = \#_L wqv$ (given δ), written $c \vdash_M c'$ for short:

1. If $a \neq \#_L$, $a \neq \#_R$ and $(q, a, L, q', a') \in \delta$ with $a' \in \tilde{\Gamma}$, then for $w = w'a$, $c = \#_L w'aqv \vdash_M \#_L w'q'a'v$ holds.

2. If $a \neq \#_L$, $a \neq \#_R$ and $(q, a, R, q', a') \in \delta$ with $a' \in \tilde{\Gamma}$, then for $w = w'a$ and $v = bv'$ with $b \in \Gamma \setminus \{\#_L\}$, $c = \#_L w'aqbv' \vdash_M \#_L wbq'v'$ holds.

3. If $a = \#_L$ and $(q, \#_L, R, q', \#_L) \in \delta$, then for $w = \lambda$ and $v = bv'$ with $b \in \Gamma \setminus \{\#_L\}$, $c = \#_L qbv' \vdash_M \#_L bq'v'$ holds.

4. If $a = \#_R$ and $(q, \#_R, L, q', a') \in \delta$ with $a' \in \tilde{\Gamma}$, then for $w = w'\#_R$, $c = \#_L w'\#_R q \vdash_M \#_L w'q'a'\#_R$ holds.

As usual, $\vdash_M^*$ denotes the reflexive transitive hull of the binary relation $\vdash_M$. The language *generated* by M is given as:

$$L(M) = \{w \in \Sigma^* \mid \#_L q_0 \#_R \vdash_M^* \#_L w\xi q_f \#_R, \text{ where } \xi \in \#^*\}.$$

Observe that only the last condition given in the definition of successor configuration allows for prolongating the working tape. This ability is, of course, essential for obtaining the power to describe all recursively enumerable languages, see also the famous workspace theorem [12]. In the following, $\mathcal{RE}$ denotes the class of all recursively enumerable languages, which can be characterized as the family of languages generatable by Turing machines.

3 Regulated grammars

The notion of a programmed grammar is crucial to this paper.

Definition 2 A *(context-free) programmed grammar (with appearance checking)* is given by a quadruple $G = (N, \Sigma, P, S)$, where N is the nonterminal alphabet, Σ is the terminal alphabet, $S \in N$ is the start symbol and P is a finite set of rules of the form $(r : A \to w, \sigma(r), \phi(r))$, where $r : A \to w$ is a context-free rewriting rule, i.e., $A \in N$ and $w \in (N \cup \Sigma)^*$ (hence, erasing rules are permitted), which is labelled by r, and $\sigma(r)$ and $\phi(r)$ are two sets of labels of such context-free rules appearing in P. $A \to w$ is termed *core rule* of $(r : A \to w, \sigma(r), \phi(r))$. $\sigma(r)$ is also called *success field* of r and $\phi(r)$ is called *failure field* of r. By $\Lambda(P)$, we denote the set of all labels of the rules appearing in P.

For $(x_1, r_1), (x_2, r_2) \in (N \cup \Sigma)^* \times \Lambda(P)$, we write $(x_1, r_1) \Rightarrow (x_2, r_2)$ iff either

$$x_1 = yAz,\ x_2 = ywz,\ (r_1 : A \to w, \sigma(r_1), \phi(r_1)) \in P, \text{ and } r_2 \in \sigma(r_1)$$

or $x_1 = x_2$, $(r_1 : A \to w, \sigma(r_1), \phi(r_1)) \in P$, A does not occur in x_1 and $r_2 \in \phi(r_1)$. Let $\overset{*}{\Rightarrow}$ denote the reflexive transitive hull of $\Rightarrow$. The language generated by G is defined as

$$L(G) = \{w \in \Sigma^* \mid (S, r_1) \overset{*}{\Rightarrow} (w, r_2) \text{ for some } r_1, r_2 \in \Lambda(P)\}.$$

The language family generated by programmed grammars is denoted by $\mathcal{PR}$. The language family generated by programmed grammars with at most k nonterminals is denoted by $\mathcal{PR}_k$.[1]

In the literature, two important variants of programmed grammars are discussed:

- In a programmed grammar $G = (N, \Sigma, P, S)$ *without appearance checking*, for every $r \in \Lambda(P)$ we have $\phi(r) = \emptyset$.
- In a programmed grammar $G = (N, \Sigma, P, S)$ *with unconditional transfer*, for every $r \in \Lambda(P)$ we have $\phi(r) = \sigma(r)$.

Dassow and Păun [4] present the following results:

[1] In any case, a number as subscript in the corresponding language class denotation will refer to a nonterminal bound for that class.

Theorem 3 $\mathcal{PR}_8 = \mathcal{PR} = \mathcal{RE}$.

Dassow and Păun pose as an open question whether or not that complexity bound could be improved.

For better distinguishability from leftmost derivations, we will call the derivation relation defined above *free derivation*.

In a *leftmost derivation of type 3*, a selected rule $(r : A \to w, \sigma(r), \phi(r))$ of a context-free programmed grammar is applied to a sential form α always in a manner choosing the leftmost occurrence of A in α for replacement. Already Rosenkrantz [20] showed for the corresponding language family $\mathcal{PR}^{\ell-3}$:

Theorem 4 $\mathcal{PR}^{\ell-3} = \mathcal{RE}$.

Let $\mathcal{PRUT}^{\ell-3}$ denote the class of languages generatable by context-free programmed grammars with unconditional transfer working under leftmost derivations of type 3. We have shown in [8]:

Theorem 5 $\mathcal{PRUT}^{\ell-3} = \mathcal{RE}$.

A derivation according to a context-free programmed grammar (without appearance checking) $G = (N, \Sigma, P, S)$ is *leftmost of type 2* if it develops as follows:

1. Start with S and apply any rule $(r : S \to x, \sigma(r), \emptyset)$ in P to S (yielding x and the *set of rule choices* $\sigma(r)$).

2. Let y be the current sentential form and $R \subseteq \Lambda(P)$ be the current set of rule choices. Then, y derives x if there are a rule label $r \in R$, where $(r : A \to \alpha, \sigma(r), \emptyset)$, and a decomposition $y = y_1 A y_2$ of y such that there is no rule $r' \in R$, where $(r' : A' \to \alpha', \sigma(r'), \emptyset)$ and A' is contained in y_1; moreoever, the string x equals $y_1 \alpha y_2$ and $\sigma(r)$ is the new set of rule choices.

3. Continue in this way until a terminal string is obtained.

A context-free programmed grammar without appearance checking working under leftmost derivation of type 2 is also known as *state grammar* [13, 17]. Due to In [4, Theorem 1.4.3], we find for the corresponding language family $\mathcal{PR}^{\ell-2}$:

Theorem 6 $\mathcal{PR}^{\ell-2} = \mathcal{RE}$.

It is also possible to define leftmost derivations of type 2 for programmed grammars with appearance checking (and hence, with unconditional transfer). Corresponding definitions can be found in [6, 8]. Due to some technicalities of the definitions (which would probably deviate from the focus of this paper), we will discuss syntactic complexity issues for these grammars only in form of remarks. We briefly mention the following result [8]:

Theorem 7 $\mathcal{PRUT}^{\ell-2} = \mathcal{RE}$.

Definition 8 A (context-free) *matrix grammar* is a quintuple $G = (N, \Sigma, M, S, F)$, where N, Σ, and S are defined as in Chomsky grammars (the alphabet of nonterminals, the terminal alphabet, and the axiom), M is a finite set of matrices each of which is a finite sequence $m : (A_1 \to w_1, A_2 \to w_2, \ldots, A_n \to w_n)$, $n \geq 1$, of context-free rewriting rules over $N \cup \Sigma$, and F is a finite set of occurrences of such rules in M. For some words x and y in $(N \cup \Sigma)^*$ and a matrix $m : (A_1 \to w_1, A_2 \to w_2, \ldots, A_n \to w_n) \in M$, we write $x \underset{m}{\Longrightarrow} y$ (or simply $x \Rightarrow y$ if there is no danger of confusion) iff there are strings $x_0, x_1, \ldots, x_n$ such that $x_0 = x$, $x_n = y$, and for $1 \leq i \leq n$, either

$$x_{i-1} = z_{i-1}A_i z'_{i-1},\ x_i = z_{i-1}w_i z'_{i-1} \text{ for some } z_{i-1},\ z'_{i-1} \in (N \cup \Sigma)^*$$

or $x_{i-1} = x_i$, the rule $A_i \to w_i$ is not applicable to x_{i-1}, and this occurrence of $A_i \to w_i$ appears in F. One says that the rules whose occurrences appear in F are used in *appearance checking mode*, and that a matrix grammar is defined with (without) appearance checking if $F \neq \emptyset$ ($F = \emptyset$). The language generated by G is defined as $L(G) = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$. The family of languages generated by context-free matrix grammars with appearance checking shall be denoted by $\mathcal{MAT}$.

In a *leftmost derivation of type 3*, a rule $A \to w$ from a matrix of a context-free matrix grammar is applied to a sential form α always in a manner choosing the leftmost occurrence of A in α for replacement.

A matrix grammar $G = (N, \Sigma, M, S, F)$ has *unconditional transfer* if F contains every occurrence of a rule. The family of languages generated by context-free matrix grammars with working under leftmost derivation of type 3 is denoted by $\mathcal{MATU}\mathcal{T}^{\ell-3}$.

If $\mathcal{MAT}$ ($\mathcal{MAT}^{\ell-3}$, respectively) denotes the class of context-free matrix languages with appearance checking working under free or leftmost derivations of type 3, respectively, then we know from [4, Theorem 4.2.3]

Theorem 9 $\mathcal{MAT}_6 = \mathcal{MAT} = \mathcal{MAT}^{\ell-3} = \mathcal{RE}$.

Furthermore, results from [6, 8] imply:

Theorem 10 $\mathcal{MATU}\mathcal{T}^{\ell-3} = \mathcal{RE}$.

Due to several technical differences and problems encountered in the definition of matrix grammars with leftmost derivations of type 2, see [3, 4, 6], we will not consider these matrix grammars here, although that it is quite clear that nonterminal complexity bounds similar to those which are derived in this paper hold for any of these grammatical mechanisms, too.

We conclude this section with discussing a simple example, namely, the language

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

and showing how to generate L economically (in terms of nonterminal complexity) by various introduced mechanisms. Observe that L cannot be generated by

any context-free programmed grammar without appearance checking using free derivations or leftmost derivations of type 3.

Example 11 $G = (\{A, B\}, \{a\}, P, A)$, where P contains the following rules:

$$\begin{array}{lll} (1 & : A \to BB, & \{1\}, \quad \{2\}), \\ (2 & : B \to A, & \{2\}, \quad \{1,3\}), \\ (3 & : A \to a, & \{3\}, \quad \{3\}). \end{array}$$

G generates L both under free derivations and under leftmost derivations of types 2 or 3.

Example 12 $G_1 = (\{A, B, F\}, \{a\}, P_1, A)$, where P contains the following rules:

$$\begin{array}{lll} (1 & : A \to BB, & \{1,2\}), \\ (2 & : A \to F, & \{3\}), \\ (3 & : B \to A, & \{3,4\}), \\ (4 & : B \to F, & \{1,5\}), \\ (5 & : A \to a, & \{5\}). \end{array}$$

G_1 is a grammar with unconditional transfer and generates L both under free derivations and under leftmost derivations of types 2 or 3.

Example 13 $G_2 = (\{A, B, Y\}, \{a\}, P_2, B)$, where P contains the following rules:

$$\begin{array}{llll} (0 & : B \to AY, & \{1,5\}, & \emptyset), \\ (1 & : A \to BB, & \{1,2\}, & \emptyset), \\ (2 & : Y \to Y, & \{3\}, & \emptyset), \\ (3 & : B \to A, & \{3,4\}, & \emptyset), \\ (4 & : Y \to Y, & \{1,5\}, & \emptyset), \\ (5 & : A \to a, & \{5,6\}, & \emptyset), \\ (6 & : Y \to \lambda, & \{6\}, & \emptyset). \end{array}$$

G_2 is a grammar without appearance checking and generates L under leftmost derivations of type 2. Note that replacements of Y only take place when all symbols have been transformed in the preceding loop due to the leftmost derivation condition.

Example 14 $G' = (\{A, B, X\}, \{a\}, M, B, F)$, where M contains the following matrices:

$$\begin{array}{l} [X \to X^4, B \to XA], \\ [X \to X^4, B \to XXXA], \\ [X \to \lambda, X \to X^4, A \to XBB, A \to A], \\ [X \to \lambda, X \to X^4, A \to XXBB, A \to X^4], \\ [X \to \lambda, X \to \lambda, X \to X^4, B \to XXA, B \to B], \\ [X \to \lambda, X \to \lambda, X \to X^4, B \to XA, B \to X^4], \\ [X \to \lambda, X \to \lambda, X \to X^4, B \to XXXA, B \to X^4], \\ [X \to \lambda, X \to \lambda, X \to \lambda, A \to XXXa], \\ [X \to \lambda, X \to \lambda, X \to \lambda, A \to X^4, B \to X^4, X \to X^4]. \end{array}$$

G' generates L both under free derivations and under leftmost derivations of type 3 when rules with right-hand side X^4 are applied in appearance checking manner. Observe how X is used to code the rule number of the corresponding programmed grammar G in unary.

Example 15 $G' = (\{A, B, X, E, E'\}, \{a\}, M, B, F)$, where M contains the following matrices:

$[X \to X^4, B \to XAEa]$,
$[X \to E, E \to \lambda, X \to X^4, A \to XBB, A \to A]$,
$[X \to E, E \to \lambda, X \to X^4, A \to XXBB, A \to X^4]$,
$[X \to E, E \to \lambda, X \to E, E \to \lambda, X \to X^4, B \to XXA, B \to B]$,
$[X \to E, E \to \lambda, X \to E, E \to \lambda, X \to X^4, B \to XA, B \to X^4]$,
$[X \to E, E \to \lambda, X \to E, E \to \lambda, X \to X^4, B \to XXXA, B \to X^4]$,
$[X \to \lambda, X \to \lambda, X \to \lambda, B \to X^4, A \to E, E \to \lambda, E \to XXXE']$,
$[X \to \lambda, X \to \lambda, X \to \lambda, B \to X^4, E \to X^4, A \to E', E' \to \lambda, E' \to XXXE'a]$,
$[X \to \lambda, X \to \lambda, X \to \lambda, A \to X^4, E \to X^4, B \to X^4, X \to X^4, E' \to \lambda]$.

G' generates L both under free derivations and under leftmost derivations of type 3 when rules with right-hand side X^4 are applied in appearance checking manner. E (and its primed version) serves as a "success witness": only if $Z \in \{E, E'\}$ is not erased from the sentential form when applying any of the matrices (besides the first and the last one), the guess of the rule labels (again, coded in unary through X) has been correct.

4 Main result

In this section, we are going to sketch a proof of the following result, thereby improving the previously known nonterminal complexity bound considerably:

Theorem 16 $\mathcal{PR}_3 = \mathcal{RE}$.

Due to Theorem 3, only the inclusion $\supseteq$ has to be shown.

4.1 Informal explanations

We proceed by giving several explanations concerning our construction on a rather intuitive level. Of course, we only need to show how to simulate a Turing machine generating some language L by a programmed grammar with three nonterminals. The three nonterminals of the simulating grammar are: A, B, and C. We consider a fixed Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_f, \#_L, \#_R, \#)$.

4.1.1 Encodings

Given a configuration $c \in (\Gamma \cup Q)^*$ of M, let $\beta(c) \in \{0,1\}^*$ denote some binary encoding of c using $\gamma = \lceil \log_2(|\Gamma| + |Q|) \rceil$ many bits per symbol from $\Gamma \cup Q$. Since

Γ contains at least three special symbols, namely, $\#_L$, $\#_R$ and $\#$, and one input symbol and since Q contains at least one symbol, we have $\gamma \geq 3$. We interpret strings $\beta(c)$ as natural numbers given in binary in a standard manner, i.e., 001 for example, would be the three-bit binary representation of the number 1. We further assume $\beta(\#) = 0^\gamma$, $\beta(\#_L) = 0^{\gamma-1}1$ and $\beta(\#_R) = 10^{\gamma-1}$. Observe that $|\beta(c)| = |c|\gamma$. Obviously, c can be codified uniquely over the unary alphabet $\{A\}$ by $A^{\beta(c)}$, which is not the empty word, because c always starts with $\#_L$.

There is a special technique which can be called "passing over symbols" which we describe next.

4.1.2 Passing over symbols

How can a tape symbol be "passed over", e.g., in the simulation phase? To this end, consider the following program fragment:

$$\begin{array}{llll} ((r,1,1): & A \to \lambda & \{(r,1,2)\} & \{(r,1,3)\} \\ ((r,1,2): & A \to C & \{(r,1,1)\} & \{(r,1,3)\} \\ ((r,1,3): & C \to A & \{(r,1,3)\} & \{(r,2,1)\} \end{array}$$

Such a program fragment is useful to transfer A^n into A^m with $m = \lfloor n/2 \rfloor$. In other words, the *rightmost* bit of $\beta(c)$ is erased. Such a loop is useful in several circumstances:

1. within the simulation loop, in order to pass over symbols which are uninteresting in the simulated step; here, one has to store the skipped symbols somehow (to this end, the symbol B is going to be used);
2. when checking the correctness of the guess on where to actually start the simulation of one Turing machine step;
3. when transforming a codified terminal string, e.g., $A^{\beta(\#_L w q_f \#_R)}$, into w with $w \in \Sigma^*$.

In each of the three described situations, the two different branches towards $(r,1,3)$ could be used to test the contents of the currently last bit of the string x stored in A^x.

4.1.3 Simulation loop

Our aim is to give several rules of the programmed grammar such that $c \vdash_M c'$ is reflected by $(A^{\beta(c)}B, r) \stackrel{*}{\Rightarrow} (A^{\beta(c')}B, r')$. We assume that the Turing machine state q of configuration c is somehow stored in the label r.

The grammar scans $A^{\beta(c)}$, searching for some codification of the subword aq, where the assumed input symbol a is guessed nondeterministically. To this end, a nondeterministically chosen number of letters is passed over, until the guessed subword aq is chosen to be verified. At the end of the verification of the correctness of the subword guess (at the chosen position), a replacement (chosen from the possibilities given by δ, hence yielding c' from c) is simulated. Finally, the intermediate representation has to be converted back into the standard codification $A^{\beta(c')}$.

4.2 A formal construction

4.2.1 Initialization

Consider

$$(\text{init} : A \to A^{\beta(\#_L q_0 \#_R)} B, \text{simstart}(q_0), \emptyset)$$

as the start rule. Here, simstart is a label set indicating possible starting points of the simulation. simstart will be defined formally below.

We already point here to one subtlety of the definition of programmed grammars: since no explicit starting rule for the grammar is given, taking A as the start symbol means that in principal, the work of the programmed grammar could start at many places. Our construction will be such that at no place other than in rule init, the code of $\#_L$ is generated. Since at the end of the simulation it is tested whether $\#_L$ is present in the string (in this case only, the simulation will possibly yield a terminal string), it it clear that one has to start any successful generation through the programmed grammar with rule init. [2]

4.2.2 Skipping a symbol

The rules for this task will have the labels (skip, q, i, j), with $q \in Q$, $1 \leq i \leq \gamma$ and $1 \leq j \leq 9$. More precisely, we take the following rules:

$((\text{skip}, q, i, 1) :$	$A \to C,$	$\{(\text{skip}, q, i, 2)\},$	$\emptyset)$
$((\text{skip}, q, i, 2) :$	$A \to A,$	$\{(\text{skip}, q, i, 3)\},$	$\emptyset)$
$((\text{skip}, q, i, 3) :$	$C \to A,$	$\{(\text{skip}, q, i, 4)\},$	$\emptyset)$
$((\text{skip}, q, i, 4) :$	$B \to C^2,$	$\{(\text{skip}, q, i, 4)\},$	$\{(\text{skip}, q, i, 5)\})$
$((\text{skip}, q, i, 5) :$	$C \to B,$	$\{(\text{skip}, q, i, 5)\},$	$\{(\text{skip}, q, i, 6)\})$
$((\text{skip}, q, i, 6) :$	$A \to \lambda,$	$\{(\text{skip}, q, i, 7)\},$	$\{(\text{skip}, q, i, 9)\})$
$((\text{skip}, q, i, 7) :$	$A \to C,$	$\{(\text{skip}, q, i, 6)\},$	$\{(\text{skip}, q, i, 8)\})$
$((\text{skip}, q, i, 8) :$	$B \to B^2,$	$\{(\text{skip}, q, i, 9)\},$	$\emptyset)$
$((\text{skip}, q, i, 9) :$	$C \to A,$	$\{(\text{skip}, q, i, 9)\},$	$\text{exit-skip}(i))$

Here, exit-skip(i) equals $\{(\text{skip}, q, i+1, 1)\}$ if $i < \gamma$ and simstart(q), otherwise.

4.2.3 Simulation

We give a separate simulation for each of the four possible cases of a Turing machine step. Fix some rule $r = (q, a, X, q', a') \in \delta$ in the following.

Case 1: $a \in \tilde{\Gamma}$ (Recall that $\tilde{\Gamma} = \Gamma \setminus \{\#_L, \#_R\}$.) and $X = L$.
Let $\beta(aq) = \beta_1 \ldots \beta_{2\gamma}$ with $\beta_i \in \{0,1\}$ and $\beta(q'a') = \beta'_1 \ldots \beta'_{2\gamma}$ with $\beta'_i \in \{0,1\}$.

[2] One could have avoided such additional complication by considering graph-controlled grammars as suggested in [5] as a possible clearer grammatical model. In fact, all results of this paper as stated for programmed grammars are also valid for these related grammatical mechanisms.

The simulation of a Turing step has two sub-phases: firstly, it it checked whether aq is codified in the current position (which has been reached by repeated applications of the skip procedure), and then $q'a'$ is generated in its place.

We take the following rules in the checking phase:

$((\text{sim-}1,r,i,1):$	$A \to C,$	$\{(\text{sim-}1,r,i,2)\},$	$\emptyset)$
$((\text{sim-}1,r,i,2):$	$A \to A,$	$\{(\text{sim-}1,r,i,3)\},$	$\emptyset)$
$((\text{sim-}1,r,i,3):$	$C \to A,$	$\{(\text{sim-}1,r,i,4)\},$	$\emptyset)$
$((\text{sim-}1,r,i,4):$	$A \to \lambda,$	$\{(\text{sim-}1,r,i,5)\},$	$f_{0,i}(\beta_{2\gamma-i+1}))$
$((\text{sim-}1,r,i,5):$	$A \to C,$	$\{(\text{sim-}1,r,i,4)\},$	$f_{1,i}(\beta_{2\gamma-i+1}))$
$((\text{sim-}1,r,i,6):$	$C \to A,$	$\{(\text{sim-}1,r,i,6)\},$	$\text{cont-sim-}1(i))$

Here,

$$f_{j,i}(b) = \begin{cases} \emptyset, & \text{if } j \neq b; \\ \{(\text{sim-}1,r,i,6)\}, & \text{if } j = b; \end{cases}$$

and

$$\text{cont-sim-}1(i) = \begin{cases} \{(\text{sim-}1,r,i+1,1)\}, & \text{if } i < 2\gamma; \\ \{(\text{sim-}1,r,1,7)\}, & \text{if } i = 2\gamma. \end{cases}$$

Moreover, we take the following rules in the generating phase:

$((\text{sim-}1,r,i,7):$	$B \to C^2,$	$\{(\text{sim-}1,r,i,7)\},$	$\{(\text{sim-}1,r,i,8)\})$
$((\text{sim-}1,r,i,8):$	$C \to B,$	$\{(\text{sim-}1,r,i,8)\},$	$f'_i(\beta'_i))$
$((\text{sim-}1,r,i,9):$	$B \to B^2,$	$\text{exit-sim-}1(i),$	$\emptyset)$

Here,

$$f'_i(b) = \begin{cases} \text{exit-sim-}1(i), & \text{if } b = 0; \\ \{(\text{sim-}1,r,i,9)\}, & \text{if } b = 1; \end{cases}$$

and $\text{exit-sim-}1(i)$ equals $\{(\text{sim-}1,r,i+1,7)\}$ if $i < 2\gamma$ and $\{(\text{return},q',1)\}$, otherwise.

In any case, we have $1 \leq i \leq 2\gamma$ and $1 \leq j \leq 9$.

Case 2: $a \in \tilde{\Gamma}$, $b \in \Gamma \setminus \{\#_L\}$ and $X = R$.
Let $\beta(aqb) = \beta_1 \ldots \beta_{3\gamma}$ with $\beta_i \in \{0,1\}$, $\beta(a'bq') = \beta'_1 \ldots \beta'_{3\gamma}$ with $\beta'_i \in \{0,1\}$.

This case can be handled completely analogously to Case 1 (only replacing coded binary subwords of length 3γ instead of 2γ). Therefore, we refer the technical construction to the reader. In this way, we get rules labelled by $(\text{sim-}2,r,b,i,j)$, with $1 \leq i \leq 3\gamma$ and $1 \leq j \leq 9$.

Case 3: $a = \#_L$, $b \in \Gamma \setminus \{\#_L\}$ and $X = R$.
Let $\beta(qb) = \beta_1 \ldots \beta_{2\gamma}$ with $\beta_i \in \{0,1\}$ and $\beta(bq') = \beta'_1 \ldots \beta'_{2\gamma}$ with $\beta'_i \in \{0,1\}$.

Again, this case is simulated quite similarly to Case 1. But since we do not want to write a codification of $\#_L$ in this place (see the remarks accompanying the initialization rule), we merely test for the occurrence of $\#_L$. Since we assume $\beta(\#_L) = 0^{\gamma-1}1$, this check can be performed quite easily. Unfortunately, the

check must be done inbetween the checking phase (for $\beta(qb)$) and the generating phase (for $\beta(bq')$). Therefore, we give a complete formal description of this case below which, with exception of the checking phase, is exactly as in Case 1 before, thus yielding rules (sim-3, r, b, i, j) with $1 \leq i \leq 2\gamma$ and $1 \leq j \leq 6$.

Next, we test for the presence of $\#_L$ at the left-hand side of the simulated Turing tape:

$((\text{sim-3}, r, b, i, 7):$	$A \to C,$	$\{(\text{sim-3}, r, b, i, 8)\},$	$\emptyset)$
$((\text{sim-3}, r, b, i, 8):$	$A \to A,$	$\emptyset,$	$\{(\text{sim-3}, r, b, i, 9)\})$
$((\text{sim-3}, r, b, i, 9):$	$C \to A,$	$\{(\text{sim-3}, r, b, i, 10)\},$	$\emptyset)$

Moreover, we take the following rules in the generating phase:

$((\text{sim-3}, r, b, i, 10):$	$B \to C^2,$	$\{(\text{sim-3}, r, b, i, 10)\},$	$\{(\text{sim-3}, r, b, i, 11)\})$
$((\text{sim-3}, r, b, i, 11):$	$C \to B,$	$\{(\text{sim-3}, r, b, i, 11)\},$	$f_i'(\beta_i'))$
$((\text{sim-3}, r, b, i, 12):$	$B \to B^2,$	$\text{exit-sim-3}(i),$	$\emptyset)$

Here,

$$f_i'(b) = \begin{cases} \text{exit-sim-3}(i), & \text{if } b = 0; \\ \{(\text{sim-3}, r, b, i, 12)\}, & \text{if } b = 1; \end{cases}$$

and exit-sim-3(i) equals $\{(\text{sim-3}, q, i+1, 10)\}$ if $i < 2\gamma$ and $\{(\text{return}, q', 1)\}$, otherwise.

In any case, we have $1 \leq i \leq 2\gamma$ and $1 \leq j \leq 12$.

Case 4: $a = \#_R$ and $X = L$.
Let $\beta(\#_R q) = \beta_1 \ldots \beta_{2\gamma}$ with $\beta_i \in \{0, 1\}$ and $\beta(q'a'\#_R) = \beta_1' \ldots \beta_{3\gamma}'$ with $\beta_i' \in \{0, 1\}$.

Such a Turing machine step mainly serves for extending the work tape. Therefore, the sentential form of the simulating grammar has to grow.

Therefore, we merely have to adapt the checking rules of Case 1, as well as the generating rules of Case 1, except for the fact that the "generating loop" has to be executed now 3γ instead of 2γ times.

In this way, we get rules labelled with (sim-4, r, i, j), with $1 \leq i \leq 2\gamma$ and $1 \leq j \leq 6$ for the checking phase, as well as with $1 \leq i \leq 3\gamma$ and $7 \leq j \leq 9$ for the generating phase.

4.2.4 Returning to standard presentation

The corresponding rules are simply obtained by interchanging the roles of A and B in the skipping construction. For $q \in Q$ and $1 \leq j \leq 10$, we take the

following rules:

$((\text{return}, q, 1):$	$B \to C,$	$\{(\text{return}, q, 2)\},$	$\emptyset)$
$((\text{return}, q, 2):$	$B \to B,$	$\{(\text{return}, q, 3)\},$	$\{(\text{return}, q, 10)\})$
$((\text{return}, q, 3):$	$C \to B,$	$\{(\text{return}, q, 4)\},$	$\emptyset)$
$((\text{return}, q, 4):$	$A \to C^2,$	$\{(\text{return}, q, 4)\},$	$\{(\text{return}, q, 5)\})$
$((\text{return}, q, 5):$	$C \to A,$	$\{(\text{return}, q, 5)\},$	$\{(\text{return}, q, 6)\})$
$((\text{return}, q, 6):$	$B \to \lambda,$	$\{(\text{return}, q, 7)\},$	$\{(\text{return}, q, 9)\})$
$((\text{return}, q, 7):$	$B \to C,$	$\{(\text{return}, q, 6)\},$	$\{(\text{return}, q, 8)\})$
$((\text{return}, q, 8):$	$A \to A^2,$	$\{(\text{return}, q, 9)\},$	$\emptyset)$
$((\text{return}, q, 9):$	$C \to B,$	$\{(\text{return}, q, 9)\},$	$\{(\text{return}, q, 1)\})$
$((\text{return}, q, 10):$	$C \to B,$	$\text{simstart}(q),$	$\emptyset)$

Here,

$$
\begin{aligned}
\text{simstart}(q) \;=\; & \{(\text{skip}, q, 1, 1)\} \\
\cup\; & \{(\text{sim-1}, r, 1, 1) \mid r \in \delta, \pi_1(r) = q, \pi_3(r) = L\} \\
\cup\; & \{(\text{sim-2}, r, b, 1, 1) \mid r \in \delta, \pi_1(r) = q, \pi_3(r) = R, b \in \Gamma \setminus \{\#_L\}\} \\
\cup\; & \{(\text{sim-3}, r, b, 1, 1) \mid r \in \delta, \pi_1(r) = q, \pi_2(r) = \#_L, \pi_3(r) = R, b \in \Gamma \setminus \{\#_L\}\} \\
\cup\; & \{(\text{sim-4}, r, 1, 1) \mid r \in \delta, \pi_1(r) = q, \pi_2(r) = \#_R, \pi_3(r) = L\} \\
\cup\; & \{(\text{term}, \#_R, 0, 0) \mid q = q_f\}
\end{aligned}
$$

Observe that only in the case when the final state has been reached, the first termination rule may be selected as the next rule to be applied after finishing a simulation loop.

4.2.5 Termination rules

Firstly, we check in some preparatory steps whether there is at least one A and exactly one B in the string. Then, we continue checking for the occurrence of $\#_R$ at the rightmost position of the simulated Turing tape.

$((\text{term}, \#_R, 0, 0):$	$A \to A,$	$\{(\text{term}, \#_R, 0, 1)\},$	$\emptyset)$
$((\text{term}, \#_R, 0, 1):$	$B \to C,$	$\{(\text{term}, \#_R, 0, 2)\},$	$\emptyset)$
$((\text{term}, \#_R, 0, 2):$	$B \to B,$	$\emptyset,$	$\{(\text{term}, \#_R, 0, 3)\})$
$((\text{term}, \#_R, 0, 3):$	$C \to B,$	$\{(\text{term}, \#_R, 1, 1)\},$	$\emptyset)$

Now, let $\hat{\Sigma} = \Sigma \cup \{\#_L, \#_R, \#, q_f\}$ be the set of symbols admissible in a configuration whose tape contains a terminal string. In addition, for $a \in \hat{\Sigma} \setminus \{\#_L\}$ with $\beta(a) = \beta_1 \ldots \beta_\gamma$, $\beta_i \in \{0, 1\}$, and for $1 \leq i < \gamma$, we have:

$((\text{term}, a, i, 1):$	$A \to \lambda,$	$\{(\text{term}, a, i, 2)\},$	$f_{0,i}(\beta_{\gamma-i+1}))$
$((\text{term}, a, i, 2):$	$A \to C,$	$\{(\text{term}, a, i, 1)\},$	$f_{1,i}(\beta_{\gamma-i+1}))$
$((\text{term}, a, i, 3):$	$C \to A,$	$\{(\text{term}, a, i, 3)\},$	$\{(\text{term}, a, i+1, 1)\})$

Here,

$$f_{j,i}(b) = \begin{cases} \emptyset, & \text{if } j \neq b; \\ \{(\text{term}, a, i, 3)\}, & \text{if } j = b. \end{cases}$$

Similarly, the first bit is finally checked:

$((\text{term}, a, \gamma, 1):$	$A \to \lambda,$	$\{(\text{term}, a, \gamma, 2)\},$	$f_{0,i}(\beta_1))$
$((\text{term}, a, \gamma, 2):$	$A \to C,$	$\{(\text{term}, a, \gamma, 1)\},$	$f_{1,i}(\beta_1))$

Then, different things may happen, depending on which tape symbol has been currently read:

$((\text{term}, \#_R, \gamma, 3):$	$C \to A,$	$\{(\text{term}, \#_R, \gamma, 3)\},$	$\{(\text{term}, q_f, 1, 1)\})$
$((\text{term}, q_f, \gamma, 3):$	$C \to A,$	$\{(\text{term}, q_f, \gamma, 3)\},$	$T(\{\#_R\}))$
$((\text{term}, \#, \gamma, 3):$	$C \to A,$	$\{(\text{term}, \#, \gamma, 3)\},$	$T(\{\#_R\}))$
$((\text{term}, \tilde{a}, \gamma, 3):$	$C \to A,$	$\{(\text{term}, \tilde{a}, \gamma, 3)\},$	$\{(\text{term}, \tilde{a}, \gamma, 4)\})$
$((\text{term}, \tilde{a}, \gamma, 4):$	$B \to \tilde{a}B,$	$T(\{\#, \#_R\}),$	$\emptyset)$

where $\tilde{a} \in \Sigma$ and $T(X) = \{(\text{term}, a, 1, 1) \mid a \in \hat{\Sigma} \setminus X\}$ for $X \subset \hat{\Sigma}$.

Finally, we check the codification of the leftmost tape symbol, i.e., $\#_L$, and yield the terminal string if everything was alright up to now.

$((\text{term}, \#_L, 1, 1):$	$A \to \lambda,$	$\{(\text{term}, \#_L, 1, 2)\},$	$\emptyset)$
$((\text{term}, \#_L, 1, 2):$	$A \to \lambda,$	$\emptyset,$	$\{(\text{term}, \#_L, 1, 3)\})$
$((\text{term}, \#_L, 1, 3):$	$B \to \lambda,$	$\{(\text{term}, \#_L, 1, 1)\},$	$\emptyset)$

4.3 The correctness of the construction

In principle, a configuration c of the simulated Turing machine (which is in state q) is codified by $A^{\beta(c)}B$ at any time before the simulation enters a rule from simstart(q). By induction, it can be shown that

1. a complete loop entering $(\text{skip}, q, 1, 1)$ and heading for some rule from simstart(q) $\setminus \{(\text{skip}, q, 1, 1)\}$ (which does not enter a rule from simstart(q) inbetween) converts a string[3] of the form $A^{\beta(wa)}B^{1(\beta(u))^R}$ into a string $A^{\beta(w)}B^{1(\beta(ua))^R}$, where $w, u \in \Gamma^*$, $a \in \Gamma$, unless $a = \#_L$ and $w = \lambda$;

2. the rules whose labels start with sim-i correctly simulate an application of a rule of type i of the Turing machine;

3. a string of the form $A^{\beta(w)}B^{1(\beta(u))^R}$ is correctly converted into a string $A^{\beta(wu)}B$ by repeated applications of rules whose labels start with return;

4. (only) a codified tape of the form $c \in \#_L w \#^* q_f \#_R$ for some $w \in \Sigma$, i.e., $A^{\beta(c)}B$ can be correctly transformed into the terminal string w;

[3]Here, B^{1x} for some $x \in \{0,1\}^*$ is the string of B's obtained by interpreting the binary string $1x$ as a binary number.

5. all sentential forms generatable by the programmed grammar are of the form $A^+(B \cup C)^+ \cup (A \cup C)^* \Sigma^* B^+ \cup \Sigma^*$.

Basically from these considerations, the correctness of the proposed construction may be inferred.

We are going to illuminate a typical run of a simulation by a simple example. Assume that

$$\#_L q_0 \#_R \vdash \#_L \#_R q_0 \vdash \#_L q_f a \#_R \vdash \#_L a q_f \#_R$$

is a terminating run of a given Turing machine. Assume further a three-bit-codification:

$$\begin{array}{lll} \beta(\#) = 000 & \beta(\#_L) = 001 & \beta(\#_R) = 100 \\ \beta(a) = 010 & \beta(q_0) = 011 & \beta(q_f) = 101 \end{array}$$

Taking binary numbers as exponents, the simulating programmed grammar derives (assuming always to describe the situation when a first labelled rule of the corresponding "subroutine" is entered):

$$\begin{array}{lcll} A & \Rightarrow & A^{1011100}B & (\text{since } \beta(\#_L q_0 \#_R) = 001011100) \\ & \stackrel{*}{\Rightarrow} & A^{1100011}B & (\text{using sim-3}) \\ & \stackrel{*}{\Rightarrow} & A^{1101010100}B & (\text{using sim-4}) \\ & \stackrel{*}{\Rightarrow} & A^{1101010}B^{1001} & (\text{using skip}) \\ & \stackrel{*}{\Rightarrow} & A^{1010101}B^{1001} & (\text{using sim-3}) \\ & \stackrel{*}{\Rightarrow} & A^{1010101100}B & (\text{using return}) \\ & \stackrel{*}{\Rightarrow} & A^{1010101}B & (\text{using term for } \#_R) \\ & \stackrel{*}{\Rightarrow} & A^{1010}B & (\text{using term for } q_f) \\ & \stackrel{*}{\Rightarrow} & A^{1}aB & (\text{using term for } a) \\ & \stackrel{*}{\Rightarrow} & a & (\text{using term for } \#_L) \end{array}$$

5 Further consequences

In this section, we are going to discuss some consequences of our main result for other grammar mechanisms. More precisely, we will consider context-free programmed grammars (with appearance checking working under leftmost derivations of type 3 and without appearance checking working under leftmost derivations of type 2)[4] and matrix grammars with appearance checking (working under free derivations and working under leftmost derivations of type 3).

5.1 Leftmost derivations

A natural variant of context-free programmed grammars with appearance checking is to consider them working under *leftmost derivations of type 3*, as was

[4]Since we mainly deal with economical characterizations of $\mathcal{RE}$ in this paper, we omit discussing leftmost derivations of type 1 here, because they characterize the context-free languages.

already done in the very first paper on programmed grammars [20]. Since the construction in our main theorem can also be viewed in a leftmost fashion, we can conclude:

Corollary 17 $\mathcal{PR}_3^{\ell-3} = \mathcal{RE}$.

Meduna and Horváth [17, Theorem 5] previously considered the nonterminal complexity of context-free programmed grammars without appearance checking working under leftmost derivations of type 2 (within Kasai's formalism of state grammars [13]). They claimed that, for these grammars, three nonterminals are enough to generate every language from $\mathcal{RE}$. Unfortunately, the coding trick used in [17, Theorem 5] does not work properly,[5] so that we consider the problem of the nonterminal complexity for context-free programmed grammars without appearance checking working under leftmost derivations of type 2 to be still open.

Nevertheless, we can conclude an upperbound of four for the nonterminal complexity of these grammars with the help of our main Theorem 16 with the help of the following remark:

Remark 1 If any possible set of rule choices of a given programmed grammar G contains only rules with the same left-hand side, then the language generated by G using leftmost derivations of type 3 equals the language generated by G using leftmost derivations of type 2.

Actually, we did not define leftmost derivations of type 2 for programmed grammars with non-empty failure fields. Since the notion of "set of rule choices" can be extended straightforwardly, the interested reader is referred to [6] for a precise definition. Due to the preceding remark, we can immediately derive:

Corollary 18 *Every recursively enumerable language can be generated by a context-free programmed grammar with appearance checking which has only three nonterminals, using leftmost derivations of type 2.*

Next, we consider programmed grammars without appearance checking working under leftmost derivations of type 2.

Theorem 19 $\mathcal{PR}_4^{\ell-2} = \mathcal{RE}$.

Proof. Due to the preceding remark, we only need to modify the construction of Theorem 16 slightly, because the premise of the remark is satisfied. The fourth nonterminal will be denoted as Y and will be used in complete analogy to Example 13.

- We take $A \to A^{\beta(\#_L q_0 \#_R)}BY$ as new initialization rule.

[5] The reader who wishes to study the proof of Meduna and Horváth should consider the possibility that a two-letter sentential form AB codified as 01001 in the simulation will yield 00101 after simulating the rule $B \to B$ (if no other applicable rule is in the present "state").

- With the exception of the last three rules of the grammar constructed in Theorem 16, i.e., except for the rules with labels $(\text{term}, \#_L, 1, i)$ with $i = 1, 2, 3$, every rule $(r : \alpha \to \beta, \sigma(r), \phi(r))$ is simulated by two rules:

  1. $((r, +) : \alpha \to \beta, \sigma(r) \times \{+, -\}, \emptyset)$ and
  2. $((r, -) : Y \to Y, \phi(r) \times \{+, -\}, \emptyset)$.

  Since Y stands at the right-hand side of the sentential form, $Y \to Y$ is only applicable if the left-hand side of rule r is not contained in the sentential form.

- The mentioned three last rules are replaced by:

$((\text{term}, \#_L, 1, 1, +) :$	$A \to \lambda,$	$\{(\text{term}, \#_L, 1, 2, \pm)\},$	$\emptyset)$
$((\text{term}, \#_L, 1, 1, -) :$	$A \to \lambda,$	$\emptyset,$	$\emptyset)$
$((\text{term}, \#_L, 1, 2, +) :$	$A \to \lambda,$	$\emptyset,$	$\emptyset)$
$((\text{term}, \#_L, 1, 2, -) :$	$Y \to \lambda,$	$\{(\text{term}, \#_L, 1, 3)\},$	$\emptyset)$
$((\text{term}, \#_L, 1, 3) :$	$B \to \lambda,$	$\{(\text{term}, \#_L, 1, 1)\},$	$\emptyset)$

Details of the construction are tedious but straightforward and, hence, omitted. □

By interpreting the simulation rules from the proof of Theorem 19 as unconditional transfer rules, we can show:

Corollary 20 *Every recursively enumerable language can be generated by a context-free programmed grammar with unconditional which has only four nonterminals, using leftmost derivations of type 2.*

5.2 Matrix grammars

Similarly, one could consider matrix languages instead of programmed languages. Due to [4, Lemma 4.1.4], matrix grammars can simulate programmed grammars at the expense of one additional nonterminal. It can be observed that the construction given in [4, Lemma 4.1.4] also works in the case of leftmost derivations of type 3. Therefore, we can state:

Corollary 21 $\mathcal{MAT}_4 = \mathcal{MAT}_4^{\ell-3} = \mathcal{RE}$.

This improves the previously published bound of 6 nonterminals for $\mathcal{MAT}$, see [4]. Recently, we were informed of a matching result for matrix grammars [9].

Remark 2 The same bound as in the previous corollary can be derived for several variants of matrix grammars with appearance checking combining different forms of leftmost and free derivation modes as elaborated in [3]. In particular, this is true when every rule is applied in leftmost-2 style, as defined in [6].

Remark 3 A simple adaptation of the simulation idea underlying the proof of Theorem 19 to matrix grammars without appearance checking working under leftmost derivation of type 2 (either as defined in [4] or as defined in [6]) yields

$$\mathcal{MAT}_5^{\ell-2} = \mathcal{RE}.$$

Here, $\mathcal{MAT}^{\ell-2}$ is the class of languages generated by matrix grammars working under leftmost derivation of type 2 (according to one of the definitions in the literature).

6 Unconditional transfer

We are now going to bound the nonterminal complexity of context-free programmed grammars with unconditional transfer working under leftmost derivations of type 3, which were shown to be computationally complete in [8] by an intrinsically non-constructive argument. The corresponding language class (for languages over the alphabet Σ) is denoted by $\mathcal{PRUT}^{\ell-3}(\Sigma)$.

To this end, recall the notion of division ordering: For $u, v \in \Sigma^*$, $u = u_1 \dots u_n$, $u_i \in \Sigma$, we say that u *divides* v, written $u|v$, if $v \in \Sigma^* u_1 \Sigma^* \dots \Sigma^* u_n \Sigma^*$. u is also called a *sparse subword* of v in this case. The famous Theorem of Higman states that every $L \subseteq \Sigma^*$ has a *finite* subset L' such that every word in L has a sparse subword in L'. If $I(u) = \{v \in \Sigma^* \mid u|v\}$ denotes the ideal of u, then Higman's Theorem gives the following presentation of L:

$$L = \bigcup_{u \in L'} (L \cap I(u)).$$

Let us call L' a *Higman basis* of L. This presentation has been one of the ideas for showing the computational completeness of $\mathcal{PRUT}^{\ell-3}$. More precisely, it is clear from our quoted construction that the nonterminal complexity of the constructed grammar basically depends on three parameters: (1) the nonterminal complexity of the simulated grammar, (2) the size of the alphabet of the language and (3) the maximal length of a word in a Higman basis of the language. This is still true when thinking about a simulation of $\mathcal{PR}^{\ell-3}$ grammars instead of starting from type-0-grammars in Kuroda normal form, as we did in [8]. We will give details of such a construction below. This means that we can consider parameter (1) as a constant due to our main theorem. Since we keep (2) fixed by definition in the following, we only need to worry about (3).

Lemma 22 *Let Σ be an alphabet. Then, there is a constant $n_{|\Sigma|}$ such that every recursively enumerable language $L \subseteq \Sigma^*$ possesses a Higman basis $\hat{L}$ such that every word $w \in \hat{L}$ obeys $|w| \leq n_{|\Sigma|}$.*

Proof. Consider a recursively enumerable universal Turing machine language

$$L_{\text{univ}}(\Sigma) = \{c(T)\$w \mid w \in L(T) \subseteq \Sigma^*\} \subset \Sigma^+\$\Sigma^*, \tag{1}$$

where c is some chosen fixed codification function for Turing machines generating languages over Σ. Let $L' \subseteq L_{\text{univ}}(\Sigma)$ be a Higman basis for $L_{\text{univ}}(\Sigma)$. By definition of $L_{\text{univ}}(\Sigma)$, $L' \subseteq \Sigma^*\$\Sigma^*$. Choose some $L \in \mathcal{RE}$ with codification c_L. Then,

$$c_L\$L = L_{\text{univ}}(\Sigma) \cap c_L\$\Sigma^* = \bigcup_{u \in L'} (L_{\text{univ}}(\Sigma) \cap I(u)) \cap c_L\$\Sigma^*.$$

Modify each $u = x\$y \in L'$ with $I(u) \cap c_L\$L \neq \emptyset$ according to the rule $u \mapsto c_L\$y$. In this way, L' is modified into

$$L'' = \{c_L\$y \mid \exists x \in \Sigma^* : x\$y \in L' \wedge I(u) \cap c_L\$L \neq \emptyset\} \subseteq c_L\$L.$$

Moreover,

$$c_L\$L = \bigcup_{u \in L''} (L_{\text{univ}}(\Sigma) \cap I(u)) \cap c_L\$\Sigma^* = \bigcup_{u \in L''} (L_{\text{univ}}(\Sigma) \cap I(u)).$$

Now, cut off the common prefix $c_L\$$ from every word in L'', thereby getting a set $\hat{L}$. We find that

$$L = \bigcup_{u \in \hat{L}} (L \cap I(u)) \quad \text{and} \quad \hat{L} \subseteq L.$$

Therefore, $\hat{L}$ is a Higman basis for L. If $n_{|\Sigma|}$ is an upperbound on the length of words in the Higman basis L' of L_{univ}, then, for every $w \in \hat{L}$, $|w| \leq n_{|\Sigma|}$. □

Now, we can conclude:

Theorem 23 $\forall \Sigma \exists c > 0 : \mathcal{PRUT}_c^{\ell-3}(\Sigma) = \mathcal{RE}$.

Proof. Consider a language $L \in \mathcal{RE}$. Let L' be a Higman basis for L. If $\lambda \in L$, then $L' = \{\lambda\}$ can be assumed, and the main difficulties in the construction in [8] can be circumvented. By introducing a success witness (as it will be done by the more involved case detailed below, see also Example 12 above) and using Theorem 16, we can show that every $L \in \mathcal{RE}$ with $\lambda \in L$ is in $\mathcal{PRUT}_4^{\ell-3}$. Let us assume $\lambda \notin L$ in the following.

For each $u \in L'$, consider $L[u] = L \cap I(u)$. Obviously, $L[u]$ is recursively enumerable. Therefore, there is a context-free programmed grammar with appearance checking $G[u] = (N, \Sigma, P, S)$ working under leftmost derivations of type 3 which generates $L[u]$. Due to Corollary 17, $|N| = 3$ may be assumed.

We will present a sequence of modifications of $G[u]$ into different but equivalent context-free programmed grammars with appearance checking $G'[u]$, $G''[u]$, $G'''[u]$ and $G^{iv}[u]$ in order to give a comparatively simple transformation of $G^{iv}[u]$ into an equivalent context-free programmed grammar with unconditional transfer.

The modified grammar $G'[u]$. It is easy to modify $G[u]$ into another context-free programmed grammar with appearance checking $G'[u] = (N', \Sigma, P', S)$ which generates $L[u]$ and obeys, in addition, that (1) $N' = N \cup \{[a] \mid a \in \Sigma\}$, (2) only rules of the form $(a : [a] \to a, \Sigma, \emptyset)$ have terminal letters at their right-hand sides and (3) $|P'| = |\Sigma| + 3$.

The modified grammar $G''[u]$. Now, consider $u = a_1 \dots a_n$. Every word $w \in L[u]$ may be decomposed as $w = w_0 a_1 w_1 \dots a_n w_n$. In other words, there are $2n+1$ easily identifiable *parts* in each word $w \in L[u]$. We say that a nonterminal A of a grammar for $L[u]$ *contributes* to parts i through j of w if (a) $i = j$ and it generates a factor of part i or if (b) $i < j$ and it generates a suffix of part i of w, all parts $i+1$ through $j-1$ and a prefix of part j of w. Modify $G'[u]$ into $G''[u] = (N'', \Sigma, P'', (S, 1, 2n+1))$ with $N'' = N' \times \{1, \dots, 2n+1\} \times \{1, \dots, 2n+1\}$. P'' contains the following rules:

- If $(r : A \to \lambda, \sigma(r), \phi(r)) \in P'$, then put

  $$((r, i, j) : (A, i, j) \to \lambda, \sigma(r) \times I \cup T, \phi(r) \times I)$$

  into P''.

- If $(r : A \to B, \sigma(r), \phi(r)) \in P'$, then put

  $$((r, i, j) : (A, i, j) \to (B, i, j), \sigma(r) \times I \cup T, \phi(r) \times I)$$

  into P''.

- If $(r : A \to BC, \sigma(r), \phi(r)) \in P'$, then put

  $$((r, i, j) : (A, i, j) \to (B, i, k)(C, k, j), \sigma(r) \times I \cup T, \phi(r) \times I)$$

  into P'' for every $i \leq k \leq j$. If $i \leq k < j$, take, in addition,

  $$((r, i, j) : (A, i, j) \to (B, i, k)(C, k+1, j), \sigma(r) \times I \cup T, \phi(r) \times I)$$

  into P''.

Here, $I = \{(i, j) \mid 1 \leq i \leq j \leq n\}$, $T = \Sigma \times \{(1, 1)\}$ and $A, B, C \in N'$. Observe that longer right-hand sides of core rules do not appear in the simulation of Theorem 16 we are referring to. Furthermore, we have terminating rules

$$((a, i, i) : ([a], i, i) \to a, \Sigma \times \{(i, i), (i+1, i+1), (i+1, i+2)\}, \emptyset)$$

for all $a \in \Sigma$ and $i = 1, \dots, 2n+1$ is odd and

$$((a_r, 2r, 2r) : (([a_r], 2r, 2r) \to a_r, \Sigma \times \{(2r+1, 2r+1)\}, \emptyset),$$

$$((a_r, 2r, 2r+1) : (([a_r], 2r, 2r+1) \to a_r, \Sigma \times \{(2r+2, 2r+2)\}, \emptyset)$$

for $1 \leq r \leq n$. Hence, it is checked that all part contributions have been correctly guessed during the derivation. Again, $L(G''[u]) = L[u]$ is obvious.

The modified grammar $G'''[u]$. The next grammar $G'''[u] = (N''', \Sigma, P''', S''')$ will have the start rule

$$S''' \to (S, 1, 2n+1)a_1 S_2 S_3 a_2 S_4 S_5 \ldots a_n S_{2n} S_{2n+1}.$$

The only modifications apply to two cases:

- Rules

  $$((r, i, j) : (A, i, j) \to (B, i, k)(C, k, j), \sigma(r) \times I \cup T, \phi(r) \times I)$$

  of $G''[u]$ with $i < k$ are replaced by the core rules $(A, i, j) \to (B, i, k)$ followed by $S_k \to \bar{S}_k(C, k, j)$ or $\bar{S}_k \to \bar{S}_k(C, k, j)$.

- Rules

  $$((a_r, 2r, 2r) : (([a_r], 2r, 2r) \to a_r, \Sigma \times \{(2r+1, 2r+1)\}, \emptyset)$$

  are replaced by the sequence of core rules $([a_r], 2r, 2r) \to \lambda$ and $\bar{S}_{2r+1} \to \lambda$. Rules

  $$((a_r, 2r, 2r+1) : (([a_r], 2r, 2r+1) \to a_r, \Sigma \times \{(2r+2, 2r+2)\}, \emptyset)$$

  are replaced by the sequence of core rules $([a_r], 2r, 2r+1) \to \lambda$ and $S_{2r+1} \to \lambda$.

The introduction of barred versions of S_i is necessary in order to prevent false partition guesses. Therefore, $L(G'''[u]) = L[u]$ is clear. Observe that the number of nonterminals of G''' is still bounded by a polynomial in n and $|\Sigma|$.

The modified grammar $G^{iv}[u]$. It is now easy to modify $G'''[u]$ into a programmmed grammar $G^{iv}[u] = (N^{iv} = N''' \cup \{F\}, \Sigma, P^{iv}, S''')$ with the (additional) property that every rule has either an empty success field or an empty failure field. Let us further assume that the right-hand side of every rule with an empty success field is replaced by a special failure symbol F.

Let $\tilde{G}^{iv}[u]$ be a "copy" of $G^{iv}[u]$ with nonterminal alphabet $\tilde{N}^{iv} = \{\langle A\rangle \mid A \in N^{iv}\}$, start rule

$$\langle S'''\rangle \to (S, 1, 2n+1)S_1 S_2 \ldots S_{2n} S_{2n+1}$$

and terminating core rules $\langle [a]\rangle \to \lambda$. Obviously, $\tilde{G}^{iv}$ derives at most the empty word. Only the structure of the grammar matters in the following. For convenience, we consider $\langle\cdot\rangle$ as a morphism from sentential forms of G^{iv} into sentential forms of $\tilde{G}^{iv}$. In particular, a core rule $A \to w$ is in P^{iv} iff $\langle A\rangle \to \langle w\rangle$ appears in $\tilde{P}^{iv}$.

The grammar $\hat{G}[u]$ with unconditional transfer. We now transform $G^{iv}[u]$ and $\tilde{G}^{iv}[u]$ into a programmed grammar

$$\hat{G}[u] = (\hat{N} = N^{iv} \cup \{E, E_1, \ldots, E_q, \hat{S}\}, \Sigma, \hat{P}, \hat{S})$$

with unconditional transfer such that $L(\hat{G}[u]) = L[u]$.

As start rule, we take $\hat{S} \to \langle S''' \rangle E S'''$. A rule $A \to w$ (of G^{iv}) with empty failure field is simulated by the sequence $\langle A \rangle \to E$, $E \to \langle w \rangle$ and $A \to w$. Here, E has the role of a success witness in the sense that E will disappear if and only if, in the simulation, rule $A \to w$ has been applied to a sentential form which does not contain A, which obviously means that the simulation was incorrect at this place. Observe that the simulation is correct, since we are dealing with leftmost derivations of type 3.

The termination phase is started by $E \to E_1 \ldots E_q$, where $q = |\Sigma|$. More precisely, let $\Sigma = \{a_1, \ldots, a_q\}$. The termination phase proceeds by looping through $[a_j] \to E_j$, $E_j \to \lambda$ for each symbol $a_j \in \Sigma$. In this way, either u is derived if an error occurred in the derivation simulation (testified by the absence of E and hence of E_j) or some word of $L[u]$ is derived by a correct simulation of G^{iv} and $\tilde{G}^{iv}$ through $\hat{G}$.

As the reader may verify, the number of nonterminals of $\hat{G}$ is bounded by a polynomial in n and $|\Sigma|$. Of course, n is bounded by the constant $n_{|\Sigma|}$ derived in the previous Lemma 22.

Concluding the construction. Finally, since $L = \bigcup_{u \in L'} L[u]$ for a suitable Higman basis L' of L and since there are no more than $|\Sigma|^{n_\Sigma}$ elements in this union, the usual construction for proving closure under union yields a programmed grammar with unconditional transfer generating L, whose number of nonterminals is bounded by a function in $|\Sigma|$. □

Admittedly, the dependence on the size of the terminal alphabet in the previous theorem appears to be somewhat peculiar and seems to be special to programmed languages with unconditional transfer. Note that the construction used in [8] entails a dependence on the minimal size of a Higman basis of a language. As the example

$$L_n = \bigcup_{i=1}^{n} \{a_i^j \mid j \geq 1\} \tag{2}$$

(with minimal Higman basis $\{a_1, \ldots, a_n\}$) shows, the size of minimal Higman bases will grow arbitrarily large with growing terminal alphabet size.

It is still an open question whether a bound on the nonterminal complexity of context-free programmed grammars with unconditional transfer working under leftmost derivation of type 3 can be derived without limiting the size of the terminal alphabet.

Since by the results of [6, Theorem 5.8] (relying on [5, Lemma 4.3]) and [8], context-free matrix grammars with unconditional transfer are also computationally complete and since the proof transforming programmed grammars

with unconditional transfer into matrix grammars with unconditional transfer can be carried out such that the number of nonterminals is only increased by a constant (namely, by using the techniques of [4, Lemma 4.1.4] and and the success witness technique employed in [8] as well as in Theorem 23), we may conclude for the corresponding language class $\mathcal{MATUT}$:

Corollary 24 $\forall \Sigma \exists c > 0 : \mathcal{MATUT}_c^{\ell-3}(\Sigma) = \mathcal{RE}$.

We finally remark that, by Example 4.1.1(iv) (which coincides with the languages defined in Eq. (2)), Dassow and Păun [4] showed that the nonterminal complexity of so-called random context grammars (with appearance checking) is not bounded by a constant. It is an interesting open question whether

$$\forall \Sigma \exists c > 0 : \mathcal{RC}_c(\Sigma) = \mathcal{RE}$$

is true, where $\mathcal{RC}$ denotes the family of languages generatable with context-free random context grammars with appearance checking.

7 Concluding discussions

In this subsection, we like to discuss whether our main Theorem 16 can be further improved.

Programmed grammars

Remark 4 It is easily seen that each language from $\mathcal{PR}_1$ is letter-equivalent to some language accepted by a finite automaton with one partially blind counter, see [10].

Alternatively and in order to keep within the notions of this paper, one can observe that the appearance checking feature is of no use if there is only one nonterminal symbol. This observation leads us to:

Lemma 25 $\{a^{2^n} \mid n \geq 0\} \in (\mathcal{PR}_2 \setminus \mathcal{PR}_1) \cup (\mathcal{PR}_2^{\ell-3} \setminus \mathcal{PR}_1^{\ell-3})$.[6]

Proof. Due to the famous result of Hauschildt and Jantzen [11], $L = \{a^{2^n} \mid n \geq 0\}$ cannot be generated by a context-free programmed grammar without appearance checking. Our above reasoning teaches us that $L \notin \mathcal{PR}_1 \cup \mathcal{PR}_1^{\ell-3}$. Due to Example 11, $L \in \mathcal{PR}_2 \cap \mathcal{PR}_2^{\ell-3}$. □

Therefore, it remains as an open question whether the inclusion $\mathcal{PR}_2 \subseteq \mathcal{RE}$ is strict or not.

Lemma 26 $\{a^{2^n} \mid n \geq 0\} \in \mathcal{PR}_3^{\ell-2} \setminus \mathcal{PR}_1^{\ell-2}$.

[6] In [4, Theorem 4.2.2], it is claimed that there exists a regular language in $\mathcal{PR}_2 \setminus \mathcal{PR}_1$, without hinting at a proof.

Proof. It is clear that $\mathcal{PR}_1^{\ell-3} = \mathcal{PR}^{\ell-2}$. Hence, $L = \{a^{2^n} \mid n \geq 0\} \notin \mathcal{PR}_1^{\ell-2}$. Example 13 shows that $L \in \mathcal{PR}_3^{\ell-2}$. □

We conjecture that $\{a^{2^n} \mid n \geq 0\} \notin \mathcal{PR}_2^{\ell-2}$.
Similarly, we can show:

Lemma 27 $\{a^{2^n} \mid n \geq 0\} \in \mathcal{PRUT}_3^{\ell-3} \setminus \mathcal{PRUT}_1^{\ell-3}$.

We conjecture that $\{a^{2^n} \mid n \geq 0\} \notin \mathcal{PRUT}_2^{\ell-3}$.

Matrix grammars

The proof we gave for Lemma 25 also applies to matrix languages[7]. Therefore, we may state:

Lemma 28 $\{a^{2^n} \mid n \geq 0\} \in \mathcal{MAT}_3 \setminus \mathcal{MAT}_1$. □

The question which of the three $\subseteq$ relations in the chain

$$\mathcal{MAT}_1 \subseteq \mathcal{MAT}_2 \subseteq \mathcal{MAT}_3 \subseteq \mathcal{MAT}_4 = \mathcal{RE}$$

is proper remains open. At least one of the first two inclusions must be strict due to the previous lemma. This already follows from [4, Theorem 4.2.4] in the case of free derivations.

Accepting grammars

We conclude this discussion by noting that it would be also of interest to discuss the nonterminal complexity of regulated grammars as language *acceptors*. This topic was initiated by [1]. Since accepting programmed grammars with appearance checking can simulate generating programmed grammars with apperance checking in a very structural way, see [2, 7], we may conclude:

Corollary 29 *For every recursively enumerable language L, there exists a context-free programmed grammar with appearance checking which accepts L.*

Similar considerations lead us to:

Corollary 30 *For every recursively enumerable language L, there exists a context-free matrix grammar with appearance checking which accepts L.*

Again, the question is whether these bounds can be improved.

Acknowledgements: We are grateful for immediate answers of our colleagues H. Bordihn and Gh. Păun concerning questions on syntactic complexity and for some discussions with F. Stephan.

[7] but not the regularity argument we provided in the preceding footnote

References

[1] H. Bordihn and H. Fernau. Accepting grammars with regulation. *International Journal of Computer Mathematics*, 53:1–18, 1994.

[2] H. Bordihn, H. Fernau and M. Holzer. Accepting Pure Grammars and Systems. Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik, Preprint Nr. 1, January 1999.

[3] J. Dassow, H. Fernau and Gh. Păun. On the leftmost derivation in matrix grammars. *International Journal of Foundations of Computer Science*, 10:61–80, 1999.

[4] J. Dassow and Gh. Păun. *Regulated Rewriting in Formal Language Theory*, volume 18 of *EATCS Monographs in Theoretical Computer Science*. Berlin: Springer, 1989.

[5] H. Fernau. Unconditional transfer in regulated rewriting. *Acta Informatica*, 34:837–857, 1997.

[6] H. Fernau. On regulated grammars under leftmost derivation. *GRAMMARS*, 3:37–62, 2000.

[7] H. Fernau and R. Freund. Accepting array grammars with control mechanisms. IN: Gh. Păun and A. Salomaa (eds.), New Trends in Formal Languages, LNCS 1218, pages 95–118, 1997.

[8] H. Fernau and F. Stephan. Characterizations of recursively enumerable languages by programmed grammars with unconditional transfer. *Journal of Automata, Languages and Combinatorics*, 4(2):117–142, 1999.

[9] R. Freund and Gh. Păun. Four-nonterminal matrix grammars characterize the family of recursively enumerable languages. Personal communication, December 2000.

[10] S. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7:311–324, 1978.

[11] D. Hauschildt and M. Jantzen. Petri net algorithms in the theory of matrix grammars. *Acta Informatica*, 31:719–728, 1994.

[12] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Reading (MA): Addison-Wesley, 1979.

[13] T. Kasai. A hierarchy between context-free and context-sensitive languages. *Journal of Computer and System Sciences*, 4:492–508, 1970.

[14] A. Meduna. Syntactic complexity of scattered context grammars. *Acta Informatica*, 32:285–298, 1995.

[15] A. Meduna. Four-nonterminal scattered context grammars characterize the family of recursively enumerable languages. *International Journal of Computer Mathematics*, 63:67–83, 1997.

[16] A. Meduna. Generative power of three-nonterminal scattered context grammars. *Theoretical Computer Science*, 246:279–284, 2000.

[17] A. Meduna and Gy. Horváth. On state grammars. *Acta Cybernetica*, 8:237–245, 1988.

[18] Gh. Păun. Six nonterminals are enough for generating each r.e. language by a matrix grammar. *International Journal of Computer Mathematics*, 15:23–37, 1984.

[19] Gh. Păun. Connections between programmed grammars and membrane computing. Personal communication, December 2000.

[20] D. J. Rosenkrantz. Programmed grammars and classes of formal languages. *Journal of the ACM*, 16(1):107–131, 1969.

[21] C. E. Shannon. A universal Turing machine with two internal states. IN: Automata Studies, pages 157–165. Princeton University Press, 1956.