Refining the Nonterminal Complexity of Graph-controlled Grammars

Henning Fernau[1,2], Rudolf Freund[3]*,

Marion Oswald[3]†, Klaus Reinhardt[1]

[1] Wilhelm-Schickard-Institut für Informatik
Universität Tübingen
Sand 13, D-72076 Tübingen, Germany
{fernau,reinhardt}@informatik.uni-tuebingen.de

[2] Also: The University of Newcastle, Newcastle, Australia
University of Hertfordshire, Hatfield, UK

[3] Institut für Computersprachen
Fakultät für Informatik
Technische Universität Wien
Favoritenstr. 9-11, A-1040 Wien, Austria
{rudi,marion}@emcc.at

Abstract

We refine the classical notion of the nonterminal complexity of graph-controlled grammars, programmed grammars, and matrix grammars by also counting, in addition, the number of nonterminal symbols that are actually used in the appearance checking mode. We prove that every recursively enumerable language can be generated by a graph-controlled grammar with only two nonterminal symbols when both symbols are used in the appearance checking mode. This result immediately implies that programmed grammars with three nonterminal symbols where two of them are used in the appearance checking mode as well as matrix grammars with three nonterminal symbols all of them used in the appearance checking mode are computationally complete. On the other hand, every unary language is recursive if it is generated by a graph-controlled grammar with an arbitrary number of nonterminal symbols but only one of the nonterminal symbols being allowed to be used in the appearance checking mode. This implies, in particular, that the result proving the computational completeness of graph-controlled grammars with two nonterminal symbols and both of them being used in the appearance checking mode is already optimal with respect to the overall number of nonterminal symbols as well as with respect to the number of nonterminal symbols used in the appearance checking mode, too.

1 Introduction

Nonterminal complexity is a classical measure of descriptional complexity of grammars. Within the area of regulated rewriting, graph-control can be seen as a framework in which many other

*Corresponding author.

†Supported by the FWF-project T225-N04.

rewriting mechanisms can be expressed, see [2, 3, 4, 14]. In particular, programmed grammars and matrix grammars can be seen as special cases of graph-controlled rewriting, see [7].

In 1984, Gheorghe Păun published a paper [12] where he showed that "six nonterminals are enough for generating each r.e. language by a matrix grammar," as already the title of the paper indicates. Up to 2001, no better bound was published in this area. Then, at the MCU conference of 2001, two independent papers [5, 8] showed that actually three nonterminal symbols are sufficient to generate each recursively enumerable language by a graph-controlled grammar. However, there were some distinctive differences in the proofs of this result:

- Fernau (also see the journal version [6]) used a Turing machine simulation to obtain his result. He actually showed that three nonterminal symbols are enough in programmed grammars (a model that is more restrictive than graph control) to generate each recursively enumerable language. The results in [8] are weaker in this respect, since they only provide an upper bound of four on the nonterminal complexity of programmed grammars.
- The simulation of Freund and Păun makes use of the universality result of register machines with two registers as obtained by Minsky [11]. Since appearance checking is only needed in the actual simulation of the register machine with its two registers, only two out of these three nonterminal symbols are ever used in appearance checking mode; the third nonterminal symbol is only of interest to compute the conversion between the concrete recursively enumerable language whose acceptance is to be simulated and the specific representation of strings used in Minsky's result. In this respect, Fernau's result was weaker, since he used all nonterminal symbols in the appearance checking mode.

With the mentioned two MCU'01 papers [5] and [8] in mind, there are a couple of natural questions regarding the optimality of the results discussed above:

- What is the computational power of graph-controlled grammars with only one or two nonterminal symbols? In other words: are the MCU'01 universality results optimal?
- What is the computational power of graph-controlled grammars with only one nonterminal symbol that actually uses appearance checking? This question is particularly interesting from the viewpoint of membrane computing, e.g., see [9], where hierarchies with respect to the number of membranes may directly depend on the number of nonterminal symbols used in the appearance checking mode in the simulated regulated grammar (graph-controlled grammar, matrix grammar).

We shall address these questions in the present paper and improve the results discussed above by showing the optimal result for graph-controlled grammars being computationally complete with only two nonterminal symbols both of them being used in the appearance checking mode, whereas, on the other hand, a unary language over a one-letter alphabet can only be recursive if generated by a graph-controlled grammars with an arbitrary number of nonterminal symbols where only one of them is allowed to be used in the appearance checking mode.

2 Definitions

For the notions from the theory of formal languages, the reader is referred to [2]. We just mention the following: An *alphabet* V is a finite non-empty set of abstract *symbols*. Given V, the free monoid generated by V under the operation of concatenation is denoted by V^*, the empty string is denoted by λ, and $V^+ := V^* - \{\lambda\}$. By $\mathbb{N}$ we denote the set of non-negative integers. $RE\,(k)$ denotes the family of recursively enumerable languages over an alphabet of cardinality k.

2.1 Register machines

A *(deterministic) register machine* is a construct $M = (m, R, l_0, l_h)$, where m is the number of registers, R is a finite set of instructions injectively labelled with elements from a given set $lab(M)$, l_0 is the initial/start label, and l_h is the final label. The instructions are of the following forms:

- $l_1 : (ADD\,(r)\,, l_2)$

  Add 1 to the contents of register r and proceed to the instruction (labelled with) l_2. (We say that we have an ADD-instruction.)

- $l_1 : (SUB\,(r)\,, l_2, l_3)$

  If register r is not empty, then subtract 1 from its contents and go to instruction l_2, otherwise proceed to instruction l_3. (We say that we have a SUBTRACT-instruction.)

- $l_h : Halt$

  Stop the machine. The final label l_h is only assigned to this instruction.

A computation of the register M starts with the instruction assigned to the initial label l_0 and eventually ends when M reaches the final label, i.e., M halts.

The results elaborated in [11] immediately lead to the following assertions:

Proposition 1 *For any recursively enumerable set L of non-negative integers there exists a register machine M with two registers accepting L in such a way that, when starting with 2^n in register 1 and 0 in register 2, M accepts the input 2^n (by halting with both registers being empty) if and only if $n \in L$. Moreover, the halting configurations are the only ones where both registers are empty.*

Proposition 2 *For any partial recursive function $f : \mathbb{N} \to \mathbb{N}$ there exists a register machine M with two registers computing f in such a way that, when starting with 2^n in register 1 and 0 in register 2, M computes $f\,(n)$ by halting with $2^{f(n)}$ in register 1 and 0 in register 2. Moreover, in no configuration both registers are empty.*

2.2 Graph-controlled grammars and programmed grammars

A *context-free graph controlled grammar* is a construct

$$G_C = (N, T, (R, L_{in}, L_{fin})\,, S)\,;$$

N and T are alphabets of non-terminal and terminal symbols, respectively, with $N \cap T = \emptyset$, $S \in N$ is the start symbol; R is a finite set of rules r of the form $(l\,(r) : p\,(l\,(r))\,, \sigma\,(l\,(r))\,, \varphi : (l\,(r)))$, where $l\,(r) \in Lab\,(G_C)$, $Lab\,(G_C)$ being a set of labels associated (in a one-to-one manner) to the rules r in R, $p\,(l\,(r))$ is a context-free production over $(N \cup T)^*$, $\sigma\,(l\,(r)) \subseteq Lab\,(G_C)$ is the *success field* of the rule r, and $\varphi\,(l\,(r))$ is the *failure field* of the rule r; $L_{in} \subseteq Lab\,(G_C)$ is the set of initial labels, and $L_{fin} \subseteq Lab\,(G_C)$ is the set of final labels. For $r = (l(r) : p\,(l\,(r))\,, \sigma\,(l\,(r))\,, \varphi\,(l\,(r)))$ and $v, w \in (N \cup T)^*$ we define $(v, l\,(r)) \Longrightarrow_{G_C} (w, k)$ if and only if

- **either** $p\,(l\,(r))$ is applicable to v, the result of the application of the production $p(l(r))$ to v is w, and $k \in \sigma\,(l\,(r))$,
- **or** $p\,(l\,(r))$ is not applicable to v, $w = v$, and $k \in \varphi\,(l\,(r))$.

The language generated by G_C is

$$\begin{aligned} L\,(G_C) = \quad \{w \in T^* \mid \quad & (w_0, l_0) \Longrightarrow_{G_C} (w_1, l_1) \Longrightarrow_{G_C} \ldots (w_k, l_k)\,,\ k \geq 1, \\ & w_j \in (N \cup T)^* \text{ and } l_j \in Lab\,(G_C) \text{ for } 0 \leq j \leq k, \\ & w_0 = S,\ w_k = w,\ l_0 \in L_{in},\ l_k \in L_{fin}\}\,. \end{aligned}$$

If the failure fields $\varphi(l(r))$ are empty for all $r \in R$, then G_C is called a *graph-controlled grammar without appearance checking.* A nonterminal symbol $A \in N$ is said to be used *in the appearance checking mode*, if there exists at least one production of the form $A \rightarrow \alpha$, $\alpha \in (N \cup T)^*$, such that for some $r \in R$ with $p(l(r)) = A \rightarrow \alpha$ the failure field $\varphi(l(r))$ is not empty.

A graph-controlled grammar $G_C = (N, T, (R, L_{in}, L_{fin}), S)$ is called a *programmed grammar* if and only if $L_{in} = L_{fin} = Lab(G_C)$. The major drawback of programmed grammars in comparison with graph-controlled grammars with respect to our goals is the absence of the possibility to specify initial rules.

2.3 Matrix grammars

A *matrix grammar* is a construct $G_M = (N, T, (M, F), S)$ where N and T are sets of *nonterminal* and *terminal symbols,* respectively, with $N \cap T = \emptyset$, $S \in N$ is the start symbol, M is a finite set of matrices, $M = \{m_i \mid 1 \leq i \leq n\}$, where the matrices m_i are sequences of the form $m_i = (m_{i,1}, \ldots, m_{i,n_i})$, $n_i \geq 1$, $1 \leq i \leq n$, and the $m_{i,j}$, $1 \leq j \leq n_i$, $1 \leq i \leq n$, are context-free productions over $N \cup T$, and F is a subset of $\bigcup_{1 \leq i \leq n,\, 1 \leq j \leq n_i} \{m_{i,j}\}$.

For $m_i = (m_{i,1}, \ldots, m_{i,n_i})$ and $v, w \in (N \cup T)^*$ we define $v \Longrightarrow_{m_i} w$ if and only if there are $w_0, w_1, \ldots, w_{n_i} \in (N \cup T)^*$ such that $w_0 = v$, $w_{n_i} = w$, and for each $j, 1 \leq j \leq n_i$,

- **either** w_j is the result of the application of $m_{i,j}$ to w_{j-1},
- **or** $m_{i,j}$ is not applicable to w_{j-1}, $w_j = w_{j-1}$, and $m_{i,j} \in F$.

The language generated by G_M is

$$\begin{aligned} L(G_M) = \{w \in T^* \mid \; & S \Longrightarrow_{m_{i_1}} w_1 \ldots \Longrightarrow_{m_{i_k}} w_k,\ w_k = w, \\ & w_j \in (N \cup T)^*,\ m_{i_j} \in M \text{ for } 1 \leq j \leq k, k \geq 1\}. \end{aligned}$$

A nonterminal symbol $A \in N$ is said to be used *in the appearance checking mode*, if there exists at least one production of the form $A \rightarrow \alpha$, $\alpha \in (N \cup T)^*$, that appears in F.

2.4 Families of languages

Let GC_{ac} / P_{ac} / M_{ac} and GC / P / M denote the families of languages that can be generated by context-free graph-controlled grammars / programmed grammars / matrix grammars with and without appearance checking, respectively. It is well known (e.g., see [2]) that GC_{ac}, P_{ac}, and M_{ac} just equal the family of recursively enumerable languages. Hence, we refine our notation and, for n, j, k with $n \geq 1$, $n \geq j \geq 0$, and $k \geq 1$, we write $GC(n, j, k)$, $P(n, j, k)$, and $M(n, j, k)$, respectively, for all languages L that can be generated by graph-controlled grammars / programmed grammars / matrix grammars in such a way that

- L is a language over some alphabet of cardinality k,
- in the graph-controlled grammars / programmed grammars / matrix grammars at most n nonterminal symbols are used,
- out of which at most j nonterminal symbols actually are used in the appearance checking mode.

2.5 Priority-multicounter-automata

To state our results, we need some notions from the areas of Petri net theory and automata theory. These concepts are reviewed in the habilitation thesis of K. Reinhardt [13].

We define a priority-multicounter-automaton by a restrictive zero-test according to an order of the counters in the following way: the first counter can be tested for zero at any time; the second counter can only be tested for zero simultaneously with the first counter; any further counter can only be tested for zero simultaneously with all preceding counters. Formally, this reads as follows:

A *priority-multicounter-automaton* is a one-way automaton described by the 6-tuple

$$A = (n, Z, \Sigma, \delta, z_0, E)$$

with the number of counters n, the set of states Z, the input alphabet Σ, the transition relation

$$\delta \subseteq (Z \times (\Sigma \cup \{\lambda\}) \times \{0, \dots, n\}) \times (Z \times \{-1, 0, 1\}^n),$$

the initial state z_0 and the accepting states $E \subseteq Z$: We consider the set of configurations

$$C_A = Z \times \Sigma^* \times \mathbb{N}^n,$$

the initial configuration

$$\sigma_A(x) = \left\langle z_0, x, \underbrace{0, \dots, 0}_{n} \right\rangle$$

for $x \in Z^*$, and the configuration transition relation

$$\langle z, ax, m_1, \dots, m_n \rangle \underset{A}{\vdash} \langle z', x, m_1 + j_1, \dots, m_n + j_n \rangle$$

if and only if $z, z' \in Z$, $a \in \Sigma \cup \{\lambda\}$, $(z, a, k; z', j_1, \dots, j_n) \in \delta$, $\forall i \leq k \ j_i = 0$.

The language recognized by a priority-multicounter-automaton A is

$$L(A) = \left\{ w \mid \exists z_e \in E \ \exists m_1, \dots, m_n \in \mathbb{N} \ \langle z_0, w, 0, \dots, 0 \rangle \overset{*}{\underset{A}{\vdash}} \langle z_e, \lambda, m_1, \dots, m_n \rangle \right\}.$$

A priority-multicounter-automaton can be modified in such a way that it has only one accepting state z_e and that all counters are empty when accepting; in that case $L(A) = \left\{ w \mid \langle z_0, w, 0, ..., 0 \rangle \overset{*}{\underset{A}{\vdash}} \langle z_e, \lambda, 0, ..., 0 \rangle \right\}$.

The family of languages over a k-letter alphabet accepted by priority-multicounter-automata with n counters of which at most j can be tested for zero in the restricted way defined above is denoted by $P_n^k CA(j)$.

As is shown in Theorem 5.6.1 in [13], the emptiness problem for a language accepted by a priority-multicounter-automaton is decidable:

Theorem 3 *The emptiness problem for priority-multicounter-automata is decidable.*

3 Results

By definition, $GC := \bigcup_{n,k \geq 1} GC(n, 0, k)$. Hauschildt and Jantzen [10] related regulated rewriting with the theory of Petri nets.[1] These links show that the (trivial) inclusion $GC \subseteq GC_{ac}$ is actually strict, exhibiting that $\{a^{2^n} \mid n \geq 0\} \notin GC$. More specifically, they showed that $\bigcup_{n \geq 1} GC(n, 0, 1)$ characterizes the unary regular languages.

In [8] it was already proved that $GC_{ac} = GC(3, 2, k)$. So the classes having remained to be investigated are $GC(2, 2, k)$ and $GC(n, 1, k)$.

We now show that two symbols, both working in the appearance checking mode, are already sufficient to obtain computational completeness:

[1] As already indicated above, also the results from [13] we use in this paper heavily rely on Petri net theory, i.e., on the decidability of Petri nets with one inhibitor arc.

Theorem 4 $GC\,(2,2,k) = RE\,(k)$.

Proof. For a given language $L \in RE\,(k)$, $L \subseteq T^*$ for some alphabet T with $card\,(T) = k$, we construct a graph-controlled grammar

$$G_C = (\{A, B\}\,, T, (R, \{i\}\,, \{f\})\,, A)$$

with $L\,(G_C) = L$ as follows:

1. Let $T = \{a_m \mid 1 \leq m \leq k\}$; then every symbol a_m in T can be interpreted as the digit m at base $k+1$; hence, every string in T^* can be encoded as a non-negative integer using the function $g_T : T^* \to \mathbb{N}$ inductively defined by $g_T\,(\lambda) = 0$, $g_T\,(a_m) = m$ for $1 \leq m \leq k$, and $g_T\,(wa) = g_T\,(w) * (k+1) + g_T\,(a)$ for $a \in T$ and $w \in T^*$. We now iteratively generate $wA^{2^{g_T(w)}}$ for some $w \in T^*$:

   Introducing a new symbol a_m and generating the corresponding number of symbols A after it mainly is accomplished by a procedure p_m generating $A^{2^{x(k+1)+m}}$ from A^{2^x}; we assume the label (m, i) to symbolize the start label of the procedure p_m and the final label of this procedure to be labelled by (m, f). As we shall show below, the actions of a register machine with two registers can be simulated by G_C with the number of copies of the two non-terminal symbols A and B representing the contents of registers 1 and 2; hence, according to Proposition 2 we need not specify the details of p_m.

   Thus, the initial phase of generating $wA^{2^{g_T(w)}}$ can be accomplished by the following rules (observe that for the procedure p_m we only specify the initial label (m, i) and the final label (m, f)):

   - $(i : A \to A, \{(m, i) \mid 1 \leq m \leq k\} \cup \{l_0\}\,, \emptyset)$;
     we may either add a new terminal symbol a_m by choosing the label (m, i) or else finish the initial phase by choosing l_0, which is the start label of the register machine to be simulated according to Proposition 1 (accepting or not $A^{2^{g_T(w)}}$);
   - $((m, f+1) : A \to B, \{(m, f+2)\}\,, \emptyset)$;
   - $((m, f+2) : B \to a_m B, \{(i, 1)\}\,, \emptyset)$;
   - $((i, 1) : A \to \lambda, \{(i, 2)\}\,, \{(i, 3)\})$;
   - $((i, 2) : B \to BB, \{(i, 1)\}\,, \emptyset)$;
   - $((i, 3) : B \to A, \{(i, 3)\}\,, \{i\})$.

   Observe that the correct sequence of terminal symbols in a sentential form of G_C is guaranteed by the sequence of rules $(m, f+1)$ (introducing the first symbol B) and $(m, f+2)$ (generating the new terminal symbol a_m) as well as $(i, 1)$ to $(i, 3)$ eliminating all symbols A possibly being before the newly generated symbol a_m and generating the corresponding number of symbols B after it, which then are renamed to symbols A again.

2. According to Proposition 1 we then simulate a register machine with two registers which halts when started with $A^{2^{g_T(w)}}$ in its first register if and only if w is in the given language L: the contents of registers 1 and 2 correspond to the numbers of non-terminal symbols A and B, respectively; the operations of incrementing and conditionally decrementing a register are performed by adding one symbol A or B or conditionally erasing one symbol A or B, respectively. As long as the current sentential form is not terminal, either symbol A or symbol B can be used for the incrementation, i.e., for each rule $l_1 : (ADD\,(r)\,, l_2)$ of the register machine, we have to specify two rules in G_C labelled by l_1 and l_1'. For each rule $l_1 : (SUB\,(r)\,, l_2, l_3)$ we only need one rule in G_C labelled by l_1. In more detail, the simulation works as follows:

- $l_1 : (ADD(1), l_2)$ is simulated in G_C by the two rules
  $(l_1 : A \rightarrow AA, \{l_2\}, \{l_1'\})$ and
  $(l_1' : B \rightarrow BA, \{l_2\}, \emptyset)$;
- $l_1 : (ADD(2), l_2)$ is simulated in G_C by the two rules
  $(l_1 : B \rightarrow BB, \{l_2\}, \{l_1'\})$ and
  $(l_1' : A \rightarrow AB, \{l_2\}, \emptyset)$;
- $l_1 : (SUB(1), l_2, l_3)$ is simulated in G_C by the rule
  $(l_1 : A \rightarrow \lambda, \{l_2\}, \{l_3\})$;
- $l_1 : (SUB(2), l_2, l_3)$ is simulated in G_C by the rule
  $(l_1 : B \rightarrow \lambda, \{l_2\}, \{l_3\})$.

3. After halting in the final label, the two registers of the register machine are empty, hence, the remaining sentential form in G_C is terminal, i.e., G_C has generated the terminal string w. On the other hand, during the whole computation in G_C at least one nonterminal symbol must be present, because the only configuration where both registers in the simulated register machine are empty occurs when the machine halts (see Proposition 1). Hence, G_C reaches the final label if and only if the underlying sentential form is terminal. This observation completes the proof. □

In programmed grammars we have to specify the starting point of a derivation, for which we use an additional nonterminal symbol that is only used in the first derivation step and never in the appearance checking mode:

Corollary 5 $P(3, 2, k) = RE(k)$.

Proof. For a given language $L \in RE(k)$, $L \subseteq T^*$ for some alphabet T with $card(T) = k$, let

$$G_C = (\{A, B\}, T, (R, \{i\}, \{f\}), A)$$

be the graph-controlled grammar with $L(G_C) = L$ as constructed in the proof of Theorem 4. Then we construct the programmed grammar

$$G_P = (\{A, B, C\}, T, (R_P, K, K), C)$$

with

- $Lab(G_P) = K = Lab(G_C) \cup \{s\}$ where s is a *new start label* not in $Lab(G_C)$;
- C is the *new start symbol* only used in the initial rule labelled by s;
- $(s : C \rightarrow A, \{i\}, \emptyset)$ is the *initial* rule;
- $R_P = R \cup \{(s : C \rightarrow A, \{i\}, \emptyset)\}$.

After the application of the new initial rule, the derivations in G_P proceed in the same way as the derivations in G_C. As a derivation in G_C stops in the final label f if and only if the underlying sentential form is terminal, we conclude $L(G_P) = L(G_C) = L$. □

The proof of the following theorem follows the proof of Theorem 3 in [8]:

Corollary 6 $M(3, 3, k) = RE(k)$.

Proof. For a given language $L \in RE(k)$, $L \subseteq T^*$ for some alphabet T with $card(T) = k$, let

$$G_C = (\{A, B\}, T, (R, \{i\}, \{f\}), A)$$

be the graph-controlled grammar with $L(G_C) = L$ as constructed in the proof of Theorem 4. Then we construct the matrix grammar

$$G_M = (\{A, B, C\}, T, (M, F), C)$$

with $L(G_M) = L(G_C) = L$ as follows:

Without loss of generality we may assume $Lab(G_C) = \{j \mid 2 \leq j \leq g-1\}$ as well as $i = 2$ and $f = g-1$; we then take

$$F = \{X \rightarrow C^g \mid X \in \{A, B, C\}\}.$$

Now denote a sequence of k equal productions p in a matrix by $(p,)^k$. Then M contains the following matrices:

(1) $(C \rightarrow A, C \rightarrow C^g, A \rightarrow CCA)$ is the start matrix;

(2) $\left((C \rightarrow \lambda,)^l C \rightarrow C^g, Y \rightarrow YC^m, X \rightarrow \alpha\right)$
for $(l : X \rightarrow \alpha, \sigma(l), \varphi(l)) \in R$ with $X, Y \in \{A, B\}$, $\alpha \in (\{A, B\} \cup T)^*$, and $m \in \sigma(l)$;

(3) $\left((C \rightarrow \lambda,)^l C \rightarrow C^g, Y \rightarrow YC^m, X \rightarrow C^g\right)$
for $(l : X \rightarrow \alpha, \sigma(l), \varphi(l)) \in R$ with $X, Y \in \{A, B\}$, $\alpha \in (\{A, B\} \cup T)^*$, and $m \in \varphi(l)$;

(4) $\left(A \rightarrow C^g, B \rightarrow C^g, (C \rightarrow \lambda,)^{g-1} C \rightarrow C^g\right)$ is the final matrix.

The start matrix (1) has to be applied as the first matrix in a successful derivation; from the start symbol C we generate one symbol A; after having applied the production $C \rightarrow C^g$ in the appearance checking mode, from this symbol A besides introducing C^2 as the encoding of the initial label 2, the start symbol A of G_C is generated by the production $A \rightarrow CCA$.

The number of symbols C encodes the label of the current rule in G_C. In every matrix of type (2) and (3), the encoding C^l of the label l, $2 \leq l \leq g-2$, is checked by eliminating exactly l symbols C in a sequence and then applying the production $C \rightarrow C^g$ in the appearance checking mode. If more than l symbols C are present in the current sentential form, then $C \rightarrow C^g$ will be applied; if less than l symbols C are present, then the derivation will stop without finishing the whole sequence of rules in the matrix thereby leaving no symbol C, but at least one nonterminal symbol A or B in the final sentential form, because according to Proposition 1 and Proposition 2 in an intermediate configuration at least one register must be non-empty which means that not both the number of nonterminal symbols A and the number of nonterminal symbols B can be zero. This symbol $Y \in \{A, B\}$ that must be present in the sentential form then also allows for the generation of the encoding of the rule m by the production $Y \rightarrow YC^m$ when the correct number l of symbols C has been present (and the right matrix with Y being present, too, has been chosen). With rules of type (2), the application of a production $X \rightarrow \alpha$ can be simulated and we proceed with a rule $m \in \sigma(l)$. With rules of type (3), the appearance checking case is covered (by introducing g symbols C in case X were present) and we proceed with a rule $m \in \varphi(l)$.

With the final matrix (4) we first check that no nonterminal symbols A and B are present anymore (which according to Proposition 1 means that we have reached the halting configuration of the register machine simulated by the graph-controlled grammar G_C); if exactly $g-1$ control symbols C are present (which encodes the final label f of G_C), after the elimination of these symbols a terminal string from L is obtained as the result of this successful computation in G_M.

The production $C \rightarrow C^g$ must not be applied in a derivation that should lead to a terminal string from L, because in every matrix at most $g-1$ symbols C can be removed, before the generation of g symbols C would be enforced again. Hence, after the first application of the production $C \to C^g$ we can never get rid of all symbols C anymore.

Finally, we observe that it does not matter at all in which sequence the nonterminal symbols A, B, and C appear in the sentential form, because for simulating a register machine by the graph-controlled grammar G_C as described in the proof of Theorem 4 only the number of nonterminal symbols A and B is relevant, and also the encoding of the label l as the number of symbols C does not depend on the position of these symbols in the sentential form. Moreover, even the simulation of the initial phase of G_C in the proof of Theorem 4 by G_M works correctly, because the construction of the simulation of G_C by G_M guarantees that the order of symbols A and B in the sentential form derived in G_M is exactly the same as in the corresponding sentential form derived in G_C. Hence, G_C is simulated by G_M in a correct way. $\square$

We now turn our attention to the families of languages that are strictly included in the family of recursively enumerable languages.

Theorem 7 $GC(1,1,k) = GC(1,0,k)$.

Proof (sketch). For each node l in the control graph which has no final label, but a No-edge leaving it (i.e., an edge leading to a node from the failure field $\varphi(l)$), we search the control graph for a path leading to a final node by using No-edges only; if such a path exists, we also re-mark l to be a final node.

The new control graph obtained by this procedure accepts the same language as the original control graph. Erasing all No-edges from the new control graph again does not change the resulting language. The main observation in this proof is the fact that as soon as we take the first No-edge in the original control graph, the underlying sentential form must already be terminal. $\square$

The following results show that even with an arbitrary number of nonterminal symbols we cannot reach computational completeness if only at most one of them can be used in the appearance checking mode.

Theorem 8 $GC(n,1,1) \subseteq P_n^1CA(1)$.

Proof. Let $G_C = (N, \{a\}, (R, L_{in}, L_{fin}), S)$ be a graph controlled grammar where $N = \{X_i \mid 1 \le i \le n\}$ and only X_1 is used in the appearance checking mode. We then construct the priority-multicounter-automaton

$$A = (n, Z, \{a\}, \delta, z_0, \{z_e\})$$

that accepts $L(G_C)$: The counters count the number of non-terminal symbols. As we have only one terminal symbol, only the number of symbols counts and we do not have to take care of the sequence of symbols in the underlying sentential form of G_C. The first counter, which is the only one that can be tested for zero, corresponds to the only non-terminal symbol that is used in the appearance checking mode. Moreover, we take the labels of G_C as the states in A.

We define the function $\delta_{n,i} : \{1, ..., n\} \to \{0,1\}$ by

$$\delta_{n,i}(j) = \begin{cases} 0 & \text{for} \quad j \neq i \\ 1 & \text{for} \quad j = i \end{cases}$$

for $1 \le i \le n$ and $n \ge 1$; for given i, $\delta_{n,i}$ can be also viewed as an n-dimensional vector. Moreover, Ω_n denotes the all-zero vector defined by $\Omega_n(j) = 0$ for $1 \le j \le n$.

In sum, we define

$$\begin{array}{rcl} A & = & \left(n, Z, \{a\}, \delta, z_0, \{z_e\}\right), \\ Z & = & Lab\left(G_C\right) \cup \{z_0, z_e\}, \\ \delta & = & \{(z_0, \lambda, 0; z, \Omega_n) \mid z \in L_{in}\} \\ & \cup & \left\{\left(z, a^{|w|_a}, 0; z', -\delta_{n,i_0} + \Sigma_{i=1}^{n} |w|_{X_i} \delta_{n,i}\right) \mid \right. \\ & & \left. \left(z : X_{i_0} \to w, \sigma(z), \varphi(z)\right) \in R, z' \in \sigma(z)\right\} \\ & \cup & \{(z, \lambda, 1; z', \Omega_n) \mid \\ & & \left(z : X_1 \to w, \sigma(z), \varphi(z)\right) \in R, z' \in \varphi(z)\} \\ & \cup & \{(z, \lambda, 0; z_e, \Omega_n) \mid z \in L_{fin}\} \end{array}$$

We should like to mention that here we use a more general variant of priority-multicounter-automata where

$$\left(z, a^{|w|_a}, 0; z', -\delta_{n,i_0} + \Sigma_{i=1}^{n} |w|_{X_i} \delta_{n,i}\right)$$

stands for a sequence of transitions starting with $(z, \lambda, 0; z_1, -\delta_{n,i_0})$, continuing with $(z_i, a, 0; z_{i+1}, \Omega_n)$ for $1 \leq i \leq |w|_a$ and finally extending $\Sigma_{i=1}^{n} |w|_{X_i} \delta_{n,i}$ into single steps where each counter is incremented by at most one. The remaining details of this construction are obvious and therefore omitted.

As can be seen from the construction of A, successful applications of productions are simulated by decrementing the counter representing the number of the symbol that appears on the left-hand side of the production, by reading as many terminal symbols as appear on the right-hand side of the production, and by adding the number of occurrences of symbols on the right-hand side of the production to the corresponding counters. On the other hand, checking for the non-appearance of the nonterminal symbol X_1 (the only nonterminal symbol that may be used in the appearance checking mode) is accomplished by testing the first counter for zero. The computation starts and ends with the states corresponding to the labels in L_{in} and L_{fin}, respectively. Hence, we conclude that a^m is accepted by A if and only if a^m can be generated by G_C. □

Corollary 9 *For every $n \geq 1$, the emptiness problem for any $GC(n,1,1)$-grammar is decidable.*

Proof. As the proof given in Theorem 8 is constructive, the decidability of the emptiness problem for any $GC(n,1,1)$-grammar directly follows from the decidability of the emptiness problem for any priority-multicounter-automaton. □

Corollary 10 *For every $n \geq 1$, every language in $GC(n,1,1)$ is recursive.*

Proof. Let $L \in GC(n,1,1)$ and $G = (N, \{a\}, (R, L_{in}, L_{fin}), S)$ be a $GC(n,1,1)$-grammar generating L; then, for each string $w \in \Sigma^*$, from G we can construct a $GC(n,1,1)$-grammar $G(w)$ such that $L(G(w))$ is empty if and only if $w \notin L$:

- $G(w) = (N, \{a\}, (R(w), L_{in}(w), L_{fin}(w)), S)$;
- for each rule $(l : X_l \to u_l, \sigma_l, \varphi_l)$ in R we take the rules
  $((l,m) : X_l \to u_l, \sigma'_l, \varphi'_l)$ in $R(w)$, $0 \leq m \leq |w| + 1$, with
  $\sigma'_l = \{(k, \min\{m + |u_l|_a, |w| + 1\}) \mid k \in \sigma_l\}$ and
  $\varphi'_l = \{(k, m) \mid k \in \varphi_l\}$;
- $L_{in}(w) = \{(l, 0) \mid l \in L_{in}\}$;
- $L_{fin}(w) = \{(l, |w|) \mid l \in L_{fin}\}$.

In fact, $G(w)$ has been constructed in such a way that

$$\begin{aligned} w \in L(G) &\Longleftrightarrow L(G(w)) = \{w\} \qquad \text{and} \\ w \notin L(G) &\Longleftrightarrow L(G(w)) = \emptyset. \end{aligned}$$

Applying Corollary 9 now completes the proof. $\square$

Corollary 11 *For every $n \geq 2$ and $k \geq 1$, the inclusion $GC(n,1,k) \subseteq GC(n,2,k)$ is strict.*

Proof. By Theorem 4, $GC(2,2,k) = RE(k)$. As any language in $GC(n,1,1)$ according to Corollary 10 is recursive, we get $GC(n,1,1) \subsetneqq GC(n,2,1)$ for every $n \geq 2$. This obviously implies $GC(n,1,k) \subsetneqq GC(n,2,k)$ for any $k \geq 2$, too. $\square$

The following result is an immediate consequence of the results established above:

Corollary 12 *The result $GC(2,2,k) = RE(k)$ is optimal with respect to the overall number of nonterminal symbols as well as with respect to the number of symbols used in the appearance checking mode, too.*

4 Conclusion

We have solved most of the open questions as described in the introduction: especially we have shown that graph-controlled grammars with only one nonterminal symbol used in the appearance checking mode are less powerful than graph-controlled grammars with at least two nonterminal symbols used in the appearance checking mode. Computational completeness can be obtained by graph-controlled grammars with only two nonterminal symbols both of them being used in the appearance checking mode, whereas for programmed grammars and matrix grammars we need one more nonterminal symbol, which only in the case of matrix grammars has to be used in the appearance checking mode, i.e., for all $m \geq 1$, $n \geq 2$, and $k \geq 1$,

$$GC(m,1,k) \subsetneqq GC(n,2,k) = RE(k) = P(n+1,2,k) = M(n+1,3,k).$$

For programmed grammars the open question remains whether or not the third nonterminal symbol (which is not used in the appearance checking mode) is needed to obtain computational completeness, whereas for matrix grammars the only open question that remains is whether or not the third nonterminal symbol is needed to be used in the appearance checking mode, because to obtain computational completeness the third nonterminal symbol is necessary anyway (see Lemma 4.2.3 in [2] referring to [1] where even for the generation of a metalinear language three nonterminal symbols are shown to be needed).

References

[1] J. Dassow and Gh. Păun. Further remarks on the complexity of regulated rewriting. Kybernetika, 21, pp. 213–227.

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

[3] H. Fernau. Graph-controlled grammars as language acceptors. Journal of Automata, Languages and Combinatorics, 2 (2) (1997), pp. 79–91.

[4] H. Fernau. Unconditional transfer in regulated rewriting. Acta Informatica, 3 (1997), pp. 837–857.

[5] H. Fernau. Nonterminal complexity of programmed grammars. In: M. Margenstern and Y. Rogozhin (editors): *Machines, Computations, and Universality; 3rd MCU.* Lecture Notes in Computer Science 2055, Springer, 2001, pp. 202–213.

[6] H. Fernau. Nonterminal complexity of programmed grammars. Theoretical Computer Science, 296 (2003), pp. 225–251.

[7] R. Freund. Asynchronous P systems on arrays and strings. In: C. Calude, E. Calude, M. Dinneen (editors): Developments in Language Theory, 8th International Conference, DLT 2004. Lecture Notes in Computer Science 3340, Springer, 2005, pp. 188–199.

[8] R. Freund and Gh. Păun. On the number of non-terminal symbols in graph-controlled, programmed and matrix grammars. In: M. Margenstern and Y. Rogozhin (editors). *Machines, Computations, and Universality. 3rd MCU.* Lecture Notes in Computer Science 2055, Springer, 2001, pp. 214–225.

[9] R. Freund and Gh. Păun. From regulated rewriting to computing with membranes: collapsing hierarchies. Theoretical Computer Science 312 (2004), pp. 143–188.

[10] D. Hauschildt and M. Jantzen. Petri net algorithms in the theory of matrix grammars. Acta Informatica, 31 (1994), pp. 719–728.

[11] M. L. Minsky. *Computation: Finite and Infinite Machines.* Prentice Hall, 1971.

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

[13] K. Reinhardt. Counting as Method, Model and Task in Theoretical Computer Science. Habilitationsschrift, Universität Tübingen, 2005.

[14] G. Rozenberg and A. K. Salomaa. Context-free grammars with graph-controlled tables. JCSS, 13 (1976), pp. 90–99.