J. Appl. Math. & Computing Vol. **22***(2006), No. 1 - 2, pp. 21 - 38*
Website: http://jamc.net

THE POWER OF PROGRAMMED GRAMMARS WITH GRAPHS FROM VARIOUS CLASSES

MADALINA BARBAIANI, CRISTINA BIBIRE, JÜRGEN DASSOW*,
AIDAN DELANEY, SZILÁRD FAZEKAS, MIHAI IONESCU,
GUANGWU LIU, ATIF LODHI, BENEDEK NAGY

Abstract. Programmed grammars, one of the most important and well investigated classes of grammars with context-free rules and a mechanism controlling the application of the rules, can be described by graphs. We investigate whether or not the restriction to special classes of graphs restricts the generative power of programmed grammars with erasing rules and without appearance checking, too. We obtain that Eulerian, Hamiltonian, planar and bipartite graphs and regular graphs of degree at least three are pr-universal in that sense that any language which can be generated by programmed grammars (with erasing rules and without appearance checking) can be obtained by programmed grammars where the underlying graph belongs to the given special class of graphs, whereas complete graphs, regular graphs of degree 2 and backbone graphs lead to proper subfamilies of the family of programmed languages.

AMS Mathematics Subject Classification : 68Q45
Key words and phrases : Programmed grammars and languages, graph controlled grammars and languages

1. Introduction

It is well-known that context-free grammars and languages are not able to describe all phenomena which occur in natural and/or programming languages. On the other hand, context-sensitive grammars and languages are very powerful

Received September 13, 2005. Revised July 19, 2006. *Corresponding author.
The first second and seventh authors are supported by FPI Fellowships from Agncia de Gesti d'Ajuts Universitaris i de Recerca (Spain) ; the second, fifth and sixth authors are grateful for the FPU Fellowships, they got from the spanish ministry; the fifth and ninth authors are supported by the grant OTKA T049409 (Hungary), the ninth author also acknowledges the financial support provided through the European Community's Human Project under contract HPRN-CT-2002-00275, SegraVis.

(up to a morphism they generate all recursively enumerable languages), and some important decision problems are undecidable for them. Therefore a lot of grammars have been introduced which only use context-free rules and have a mechanism which controls the derivation. One of the most important and well investigated classes of such grammars is formed by the programmed grammars introduced by Rosenkrantz in 1969 ([12]).

In a programmed grammar with appearance checking, with any context-free rule $p = A \to w$ two sets of rules, the success and the failure field of p, are associated. If the rule p has to be applied, two situations can occur: if the nonterminal A occurs in the sentential form, we apply the rule $A \to w$ and continue with a rule of the success field; in the other case we do not change the sentential form and continue with a rule chosen from the failure field. If all failure fields are empty, we say that the grammar is without appearance checking.

It has been shown that programmed grammars with erasing rules and appearance checking are as powerful as arbitrary phrase structure grammars (type-0 grammars). On the other hand, if the appearance checking feature is not taken into consideration one obtains a proper subfamily of the family of recursively enumerable languages. For results on programmed grammars and languages we refer to [5], [6], [2], the references mentioned in these books and articles and the recent papers [8] and [9].

Obviously, a programmed grammar without appearance checking can be described by a graph, where the vertices are given by the rules and the edges are given by the success field, i.e., there is an edge from a rule p to a rule q in the graph if and only if q is in the success field of p.

In this paper we study the generative power of programmed grammars, where we require that the associated graph belongs to a class of special graphs. We motivate this problem as follows:

- In complexity theory the restriction to special graphs can change the situation drastically. For instance, the problem whether or not a given arbitrary graph can be coloured with 4 colours is NP-complete; however, for planar graphs the problem is trivial since any planar graph can be coloured with 4 colours.
- The restriction to special graphs has already been discussed for grammars controlled by a bicoloured directed graph (see [5]), which are equivalent to programmed grammars with appearance checking. It has been shown by Pascu and Păun that any language which can be generated by a grammar controlled by a bicoloured directed graph can be generated by a grammar controlled by a planar bicoloured directed graph (see [11] and [5], Lemma 2.2.3), i.e., the restriction to planar graphs do not decrease the generative power.
- If one considers the transformation of a matrix grammar (without appearance checking) into a programmed grammar (see [6], Proof of Theorem 2.4), one gets programmed grammars with graphs of a very special

form (union of some line graphs and two additional nodes n_1 and n_2, where, additionally, the final node of any line graph is connected with the initial node of any line graph and with n_2 and n_1 is connected with the initial node of any line graph). Thus programmed grammars with graphs of this special type have the same power as programmed grammars. However, this class of special graphs is not investigated in graph theory. In this paper we shall consider some classes of graphs which are well-known in the theory of graphs or networks.

- There exist already some papers which study restrictions with respect to the control mechanism. For instance, programmed grammars are equivalent to regularly controlled grammars. In the paper [1] it has been shown that the restriction to special classes of regular languages, e.g. to non-counting or suffix-closed regular languages, do not decrease the generative power whereas the restrictions to definite languages restricts the power to the generation of only context-free languages. Analogous results are obtained in [4] and [3] for other control mechanisms using regular languages.

In this paper we study the generative power of programmed grammars, where we require that the associated graph has certain properties. We obtain that connected, Eulerian, Hamiltonian, planar and bipartite graphs and regular graphs of degree at least three are pr-universal in the sense that all languages which can be generated by programmed grammars (with erasing rules and without appearance checking) can be obtained by programmed grammars where the underlying graph has the given special form. On the other hand, complete graphs, regular graphs of degree 2 and backbone graphs generate proper subfamilies of the family of programmed languages.

The paper is organized as follows. In the following section we present the notation used in the paper. In Section 3 we show the pr-universality of the classes of connected, Hamiltonian, Eulerian, planar and bipartite graphs. In Section 4 we study graphs with a restricted degree. In Section 5 we discuss the power of complete graphs and backbones. In the final Section 6 we discuss the situation for some classes of grammars nearly related to programmed grammars with erasing rules and without appearance checking.

2. Definitions

Throughout the paper we assume that the reader is familiar with the basic notions of the theory of formal grammars and languages including programmed grammars and languages as well as those of the theory of graphs. For details we refer to [10], [13], [5], [6], [2] and [7]. In this section we only recall some concepts and give some notations.

For an alphabet T, a word $w \in T^*$ and a letter $a \in T$, we denote the number of occurrences of a in w by $\#_a(w)$. The empty word is denoted by λ.

A context-free grammar is specified as a quadruple $G = (N, T, S, P)$, where N is a finite non-empty set called the nonterminal alphabet, T is a finite non-empty set called the terminal alphabet ($N \cap T = \emptyset$), $S \in N$ is the start symbol, and P is a finite subset of $N \times (N \cup T)^*$ called the set of rules. Rules are also named as productions.

Let G be a context-free grammar and $v, w \in (N \cup T)^*$. Then $v \Longrightarrow w$ is a direct derivation if and only if there exist $v_1, v_2, w' \in (N \cup T)^*$ and $A \in N$ such that $v = v_1 A v_2$, $w = v_1 w' v_2$ and $A \to w' \in P$.

A programmed grammar (without appearance checking) is a six-tuple $G = (N, T, S, Lab, P, P_G)$ where N, T and S are specified as in a context-free grammar, Lab is an alphabet (of labels), P is a finite set of context-free rules called the set core productions, and P_G is a finite set of triples $r = (q, p, \sigma)$, where $q \in Lab$ is the label of r, $p \in P$ is a context-free production called the core production of r, and σ is a subset of Lab and is termed the success field of r. The elements of P_G are called the rules of G.

The language $L(G)$ generated by a programmed grammar G specified as above is defined as the set of all words $w \in T^*$ such that there is a derivation

$$S = w_0 \Longrightarrow_{r_1} w_1 \Longrightarrow_{r_2} w_2 \Longrightarrow_{r_3} \ldots \Longrightarrow_{r_k} w_k = w,$$

where $k \geq 1$ and, for $1 \leq i \leq k$, $w_{i-1} = w^{'}_{i-1} A_i w^{''}_{i-1}$ and $w_i = w^{'}_{i-1} v_i w^{''}_{i-1}$ for some words $w^{'}_{i-1}, w^{''}_{i-1} \in (N \cup T)^*$, $r_i = (q_i, A_i \to \; v_i, \sigma_i)$ and, for $i < k$, $q_{i+1} \in \sigma_i$.

We say that the programmed grammar $G = (N, T, S, Lab, P, P_G)$ is in normal form if and only if $r = (q, p, \sigma) \in P_G$ and $r' = (q', p, \sigma') \in P_G$ imply $\sigma = \sigma'$. We note that, for any language L, which can be generated by a programmed grammar, there is a programmed grammar G' in normal form such that $L = L(G')$. This can be seen as follows: Assume that $L = L(G)$ for some programmed grammar $G = (N, T, S, Lab, P, P_G)$. If P_G contains two rules $r = (q, A \to w, \sigma)$ and $r' = (q', A \to w, \sigma')$ with $\sigma \neq \sigma'$, then we construct
– $P'_G = P_G \setminus \{r'\}$ (we cancel r'),
– $P''_G = P'_G \cup \{r' = (q', A \to A', \{q''\}), r'' = (q'', A' \to w, \sigma')\}$ and
– $G'' = (N \cup \{A'\}, T, S, Lab \cup \{q''\}, P \cup \{A \to A'\}, P''_G)$,
where A' is a new symbol and q'' is a new label.

For any derivation $uAv \Longrightarrow_{r'} uwv$ in G there is a derivation $uAv \Longrightarrow_{r'} uA'v \Longrightarrow_{r''} uwv$ in G'', and conversely. Thus one has $L(G) = L(G'')$. Moreover, every rule $A \to w$ occurs less times in the second components of rules of G'' than of G. By repetition of this construction we obtain a programmed grammar in normal form.

By this normal form result we can assume that there is a one-to-one function from Lab onto P.

A (finite directed) graph H is specified as a pair (V, E), where V is a finite set and $E \subseteq \{(\alpha, \beta) | \alpha, \beta \in V\}$. The elements of V and E are called vertices and edges, respectively. (α, β) is called an edge from α to β. We note that we have no multiple edges since E is a set. A graphic representation of a graph can be given as follows: We interpret the vertices as points in a plane, and we draw a (directed) curve from α to β if there is an edge (α, β).

We state some basic definitions for graphs. We say that the vertices α and β are incident with the edge $e \in E$ if $e = (\alpha, \beta)$. The *in-degree* of a vertex α, denoted by $d_i(\alpha)$, is the number of edges (β, α), and the *out-degree* $d_o(\alpha)$ is the number of edges (α, β); the *degree* $d(\alpha)$ is the sum of the in-degree and out-degree of α.

A *walk* is an alternating sequence of vertices and edges, with each edge being incident to the vertices immediately preceding and succeeding it in the sequence, and which starts and ends with vertices, which are called the initial and terminal vertices, respectively. A walk is *closed* if the initial vertex is also the terminal vertex. A *trail* is a walk with no repeated edges. A *cycle* is a closed trail with at least one edge and with no repeated vertices except that the initial vertex is the terminal vertex. A *path* is a walk with no repeated vertices. A *line* is a non-closed path which is a trail, too.

A graph is called a cycle or a line if it consists exactly of a cycle or a line, respectively.

We now define some classes of graphs.

Connected graph: A graph $G = (V, E)$ is called connected if and only if, for every two different vertices α and β there is a walk with starting vertex α and terminating vertex β or there is a walk with starting vertex β and terminating vertex α.

Hamiltonian graph: A graph $G = (V, E)$ is called Hamiltonian if it contains a cycle which contains any vertex of V.

Eulerian graph: A graph $G = (V, E)$ is said to be an Eulerian graph if it contains a path which contains every edge of E.

Bipartite graph: A graph $G = (V, E)$ is called a k-partite graph if the set V of vertices can be decomposed into k disjoint non-empty sets such that, for each edge (α, β), α and β belong to different sets of the decomposition. A bipartite graph is a special case of a k-partite graph with $k = 2$.

Planar graph: A graph is called a planar graph if its graphic representation can be drawn in the plane in such a way that there are no "edge crossings" (i.e., edges intersect only at common vertices).

Regular graph: A graph $G = (V, E)$ is called regular of degree n, if $d(\alpha) = n$ holds for any vertex $\alpha \in V$.

Complete graph: A graph is called a complete graph if, for every two vertices α and β, there is an edge from α to β. (Note that we allow $\alpha = \beta$.)

Backbone graph: A graph $G = (V, E)$ is called a *backbone graph* if the following conditions are satisfied:
– G is connected,
– there are subsets $V' \subseteq V$ and $E' \subseteq E$ such that (V', E') is a cycle,

– for any subsets $V'' \subseteq V$ and $E'' \subseteq E$ such that $(V'', E'') \neq (V', E')$, (V'', E'') is not a cycle.

Whereas the first seven types of graphs are well-known and intensively studied in the theory of graphs, the last type is motivated by computer networks which often have the structure of an (undirected) backbone graph.

We now give a description of a programmed grammar by a graph.

Let $G = (N, T, S, Lab, P, P_G)$ be a programmed grammar. We construct the graph $h(G) = (Lab, E)$, where $(q, q') \in E$ if and only if $(q, p, \sigma) \in P_G$, $(q', p', \sigma') \in P_G$ and $q' \in \sigma$. We say that G is a programmed grammar with graph $h(G)$. If G is in normal form, we can identify Lab and P, and therefore we sometimes write $h(G) = (P, E')$ where $(p, p') \in E'$ if and only if $(q, q') \in E$ and $(q, p, \sigma), (q', p', \sigma') \in P_G$.

Conversely, given a graph $H = (V, E)$, a set P of context-free productions over some sets N of nonterminals and T of terminals, and a mapping $\tau : V \to P$, we construct the programmed grammar $g(H, S) = (N, T, S, V, P, P_G)$ where $P_G = \{(q, \tau(q), \{q' \mid (q, q') \in E\}) \mid q \in V\}$ and $S \in N$. If we define or draw the graph $H = (V, E)$ equipped with a mapping τ as above, then we often give the label $\tau(v)$ instead of v.

In both cases,

$$S \Rightarrow_{r_1} w_1 \Rightarrow_{r_2} w_2 \ldots \Rightarrow_{r_k} w_k = w$$

is a derivation in G if and only if $r_1 r_2 \ldots r_k$ is a sequence of rules according to vertices along a walk in H. Control in programmed grammars corresponds to control by vertex sequences along walks in graphs. Thus, for any programmed grammar $G = (N, T, S, Lab, P, P_G)$ and any graph H with vertices labelled by (context-free) rules, we have

$$g(h(G), S) = G \text{ and } h(g(H, S)) = H\,. \tag{1}$$

Example 1. We consider the programmed grammar

$$G = (\{S, A, B\}, \{a, b, c\}, S, \{q_1, q_2, q_3, q_4, q_5\}, P, \{r_1, r_2, r_3, r_4, r_5\})$$

with

$$P = \{S \to AB, A \to aAb, B \to cB, A \to ab, B \to c\}$$

and

$r_1 = (q_1, S \to AB, \{q_2\})$, $\quad r_2 = (q_2, A \to aAb, \{q_3\})$, $\quad r_3 = (q_3, B \to cB, \{q_2, q_4\})$,
$r_4 = (q_4, A \to ab, \{q_5\})$, $\quad r_5 = (q_5, B \to c, \emptyset)$.

Then the associated graph $h(G)$ is

$$\begin{tikzcd} q_1 \arrow[r] & q_2 \arrow[r, bend left] & q_3 \arrow[l, bend left] \arrow[r] & q_4 \arrow[r] & q_5 \end{tikzcd}$$

and the generated language is $L(G) = \{a^n b^n c^n \mid n \geq 2\}$.

Let $\mathcal{H}$ be a class of graphs. We say that $\mathcal{H}$ is *pr-universal*, if and only if, for any programmed grammar G (with erasing rules and without appearance checking), there is a programmed grammar G' such that $h(G') \in \mathcal{H}$ and $L(G) = L(G')$.

3. Some pr-universal classes of graphs

In this section we present some classes of graphs such that any programmed language can be obtained from a programmed grammar where the associated graph belongs to the class.

We start with a simple statement.

Theorem 1. *Any language generated by a programmed grammar can be generated by a programmed grammar with a connected graph.*

Proof. We consider the programmed grammar $G = (N, T, S, Lab, P, P_G)$ with graph $h(G) = (P, E)$ as defined in Section 2. Let F be a new symbol not contained in N. Then we construct the graph

$$H = (P \cup \{F \to F\}, E \cup \{(F \to F, p) \mid p \in P_G\} \cup \{(p, F \to F) \mid p \in P_G\})$$

and the programmed grammar $g(H, S)$. Obviously, H is connected. Moreover, it is easy to see that $L(g(H, S)) = L(G)$ (because $F \to F$ cannot be applied in a terminating derivation). □

Theorem 2. *Any language generated by a programmed grammar can be generated by a programmed grammar with a Hamiltonian graph.*

Proof. We present a constructive proof. Let $L = L(G)$ for some programmed grammar $G = (N, T, S, Lab, P, P_G)$ in normal form with the associated graph $h(G) = (\{\alpha_1, \alpha_2, \ldots, \alpha_n\}, E)$, where n is the cardinality of P_G. We introduce new nonterminals F_i, $1 \leq i \leq n$, which are not in N. We now take n new vertices β_i with rule $S \to F_i$. Then we construct the graph

$$\begin{aligned} H &= (\{\alpha_1, \alpha_2, \ldots, \alpha_n, \beta_1, \beta_2, \ldots, \beta_n\}, \\ &\qquad E \cup \{(\alpha_i, \beta_i) \mid 1 \leq i \leq n\} \cup \{(\beta_i, \alpha_{i+1}) \mid 1 \leq i < n\} \cup \{(\beta_n, \alpha_1)\}\,. \end{aligned}$$

Obviously, H is a Hamiltonian graph since the added vertices and edges result in the (closed) Hamiltonian trail

$$\alpha_1(\alpha_1, \beta_1)\beta_1(\beta_1, \alpha_2)\alpha_2(\alpha_2, \beta_2)\beta_2(\beta_2, \alpha_3) \ldots (\alpha_n, \beta_n)\beta_n(\beta_n, \alpha_1)\alpha_1\,.$$

¿From H we construct a programmed grammar $g(H, S)$ as mentioned in Section 2. It is easy to see that $g(H, S)$ and G only differ in the rules $S \to F_i$, $1 \leq i \leq n$, their associated success fields and the adding of $S \to F_i$ to the success field of the rule associated with the vertex α_i. Because the added rules are not applicable (if S does not occur in the current sentential form) or its application introduces a letter F_i, $1 \leq i \leq n$, which cannot be terminated (since there is no rule with F_i on its left hand-side), we cannot apply these rules in terminating derivations. Therefore we can only choose rules of the success field in $g(H, S)$ which can be chosen in G, too. Thus $L(g(H, S)) = L(G)$. □

Now we take a look at the language family generated by programmed grammars with Eulerian graphs. There is a well known, nice and simple characterization of Eulerian graphs (see [7]) which is recalled in the following lemma.

Lemma 1. *A directed connected graph $H = (V, E)$ is Eulerian if and only if one of the following two possibilities hold:*
– the in-degree of every vertex equals its out-degree, or
– there are two vertices α and β with $d_o(\alpha) = d_i(\alpha) + 1$ and $d_i(\beta) = d_o(\beta) + 1$ and $d_i(\gamma) = d_o(\gamma)$ for all $\gamma \in V \setminus \{\alpha, \beta\}$.

Theorem 3. *Any language generated by a programmed grammar can be generated by a programmed grammar with an Eulerian graph.*

Proof. Let $L = L(G)$ for some programmed grammar $G = (N, T, S, Lab, P, P_G)$. Without loss of generality we can assume that G is in normal form. Further, by Theorem 1 we can assume that G is a programmed grammar with a connected graph $h(G) = (\{\alpha_1, \alpha_2, \ldots, \alpha_n\}, E)$.

We introduce new vertices $\beta_1, ..., \beta_n$, where n is the number of vertices in $h(G)$ and assign to β_i the rule $S \to F_i$, $1 \leq i \leq n$, where F_1, F_2, … , F_n are new nonterminals which cannot be terminated since there are no rules for them. We add edges to E in the following way:
– there is an edge from α_i to the new vertex β_j if and only if there is no edge from α_i to α_j in E, and
– there is an edge from from β_i to α_j if and only if there is no edge from α_i to α_j in E.
By this construction any vertex α_i, $1 \leq i \leq n$, has the in-degree n and the out-degree n. Consequently, for $1 \leq i \leq n$, in-degree and out-degree of α_i coincide.

Let us consider the vertices β_i, $1 \leq i \leq n$. If the in-degree and out-degree coincide for any β_i, then the obtained graph is Eulerian by Lemma 1. Now assume, that there is a vertex β_r such that $d_i(\beta_r) > d_o(\beta_r)$. We note that

$$\sum_{i=1}^{n} (d_i(\alpha_i) + d_i(\beta_i)) = \sum_{i=1}^{n} (d_o(\alpha_i) + d_o(\beta_i))$$

because any edge is counted as input edge as well as output edge. Since the in-degree and out-degree coincide for any vertex α_i, there is a vertex β_s such that $d_i(\beta_s) < d_o(\beta_s)$. We now connect these two vertices by an edge from β_r to β_s. Obviously, the difference of in-degree and out-degree of β_r is decreased by 1 by this procedure, and that of out-degree and in-degree of β_s, too. Continuing this construction we finally get a graph H where the in-degree and out-degree are equal for any vertex.

Let $g(H, S)$ be the programmed grammar obtained from H by the construction given in Section 2. It is easy to see that $L(G) = L(g(H, S))$ since the application of a rule associated with an added vertex yields a non-terminating derivation. □

Theorem 4. *Every language generated by a programmed grammar can be generated by a programmed grammar with a planar graph.*

Proof. Let a programmed grammar $G = (N, T, S, Lab, P, P_G)$ with graph $h(G)$ be given. If $h(G)$ is a planar graph, we are done. In the other case, $h(G)$ contains at least a crossing of edges. Below we present a procedure to substitute a crossing by a planar subgraph without changing the language and without producing new crossings. By an iterated application of this construction we finally get a planar graph.

We now consider a crossing of two edges of $h(G)$ as shown in Figure 1. The

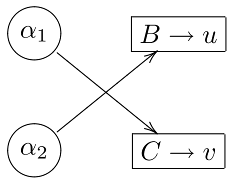

FIGURE 1. Crossing edges

rules associated with α_1 and α_2 are not of importance for our construction. Our idea is to replace the crossing point by a vertex; however, then coming from α_1 we can continue with $B \to u$ or $C \to v$ whereas in the given graph we have to continue with $C \to v$. Therefore we introduce four further vertices which control the derivation such that only the derivations of the original graph/grammar are non-blocked derivations. Precisely, we use the graph presented in Figure 2.

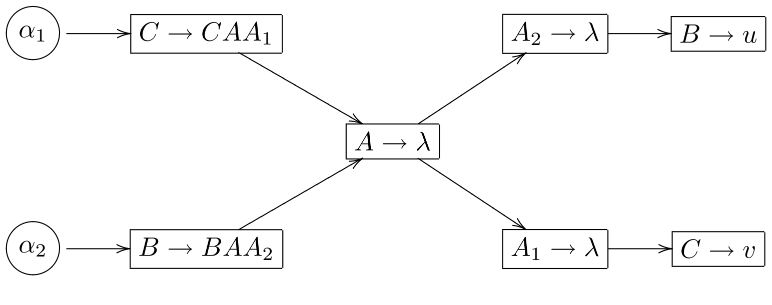

FIGURE 2. Substitution graph for crossing edges, where A, A_1 and A_2 are new nonterminals associated with the considered crossing

We now construct a new graph H_1; we substitute the graph shown in Figure 1 by the graph shown in Figure 2. Obviously, H_1 contains one crossing less than $h(G)$. Moreover, if we apply the rule of α_1 and continue with $C \to v$ according to Figure 1, then we have to apply $C \to CAA_1$ and $A \to \lambda$ according to Figure 2.

A continuation with $A_2 \to \lambda$ is impossible, since the sentential form does not contain A_2. Therefore the only non-blocked continuation is an application of $A_1 \to \lambda$ which results in the same sentential form as after the application of the rule of α_1 und we continue with $C \to v$ as in Figure 1. Thus the programmed grammar $g(H_1, S)$ constructed from H_1 and G generate the same language. □

Theorem 5. *Every language generated by a programmed grammar can be generated by a programmed grammar with a bipartite graph.*

Proof. For a given programmed grammar $G = (N, T, S, Lab, P, P_G)$ in normal form, we construct a programmed grammar G' with a bipartite graph which generates the same language as G.

Let $r = (q, p, \sigma)$ be a rule of P_G. We set

$$N_r = \{A \mid A \in N,\ (q', A \to w, \sigma') \in P_G, q' \in \sigma\}\,.$$

With r and $A \in N_r$ we associate the new nonterminals $A_r^{(1)}$ and $A_r^{(2)}$, the new labels q_r and $q_{A,r,i}$ with $A \in N_r$ and $1 \leq i \leq 3$, and the rules

$$\begin{aligned}
t_r &= (q_r, p, \{q_{A,r,1} \mid A \in N_r\}),\\
t_{A,r,1} &= (q_{A,r,1}, A \to A_r^{(1)}, \{q_{A,r,2}\}),\\
t_{A,r,2} &= (q_{A,r,2}, A_r^{(1)} \to A_r^{(2)}, \{q_{A,r,3}\}),\\
t_{A,r,3} &= (q_{A,r,3}, A_r^{(2)} \to A, \{q_s \mid s = (q'', A \to w, \sigma'') \in P_G, q'' \in \sigma,\ A \in N_r\}).
\end{aligned}$$

Moreover, we introduce a further new nonterminal S', a new label q' and the rule

$$r' = (q', S' \to S, \{q_s \mid s = (l, S \to u, \sigma''') \in P_G\}.$$

We now define the programmed grammar $G' = (N', T, S', Lab', P', P_{G'})$ by

$$\begin{aligned}
N' &= N \cup \{A_r^{(1)} \mid r \in P,\ A \in N_r\} \cup \{A_r^{(2)} \mid r \in P,\ A \in N_r\},\\
Lab' &= Lab_1 \cup Lab_2,\\
Lab_1 &= \{q'\} \cup \{q_{A,r,1} \mid r \in P,\ A \in N_r\} \cup \{q_{A,r,3} \mid r \in P_G,\ A \in N_r\}\\
Lab_2 &= \{q_r \mid r \in P_G\} \cup \{q_{A,r,2} \mid r \in P_G,\ A \in N_r\},\\
P' &= P \cup \{S' \to S\} \cup \bigcup_{\substack{r \in P_G\\ A \in N_r}} \{A \to A_r^{(1)}, A_r^{(1)} \to A_r^{(2)}, A_r^{(2)} \to A\},\\
P_{G'} &= \{r\} \cup \bigcup_{\substack{r \in P_G\\ A \in N_r}} \{t_r, t_{A,r,1}, t_{A,r,2}, t_{A,r,3}\}
\end{aligned}$$

We remark that there is a one-to-one-relation between Lab' and $P_{G'}$.

First we note that any derivation in G' starts with $S' \Longrightarrow S$ and we have to apply a rule p, where the core production of p has the left-hand side S. Moreover, there is a derivation

$$z_1 \Longrightarrow_r z_2 A z_2' \Longrightarrow_{r'} z_2 w z_2'$$

in G, where $r = (q, p, \sigma)$, $r' = (q', A \to w, \sigma') \in P_G$ and $q' \in \sigma$, if and only if there is a derivation

$$z_1 \Longrightarrow_{t_r} z_2 A z_2' \Longrightarrow_{t_{A,r,1}} z_2 A_r^{(1)} z_2' \Longrightarrow_{t_{A,r,2}} z_2 A_r^{(2)} z_2' \Longrightarrow_{t_{A,r,3}} z_2 A z_2' \Longrightarrow_{t_{r'}} z_2 w z_2'$$

in G'. This implies $L(G') = L(G)$.

Furthermore, the graph $h(G')$ associated with G' is bipartite with the partition of Lab' into Lab_1 and Lab_2, because the success field of any rule with a label of Lab_1 is contained in Lab_2 and the success field of any rule with a label of Lab_2 is contained in Lab_1. □

4. Graphs with restriction of degrees

In this part we are dealing with graphs, whose vertices have degree at most n or exactly n where $n \in \mathbb{N}$.

We discuss first the situation where $n = 2$. Connected graphs where each vertex has a degree at most two either form a cycle or any subgraph H_a consisting of all nodes reachable from the node a (by a directed path) forms a line.

Lemma 2. i) *Given a programmed grammar whose corresponding graph is a directed cycle, then its generated language is finite.*

ii) *Any finite language can be generated by a programmed grammar whose corresponding graph is a cycle.*

Proof. i) Assume that there exists a sentential form which is obtained by a derivation with more intermediate steps than the number of vertices in the cycle. This implies that at least one nonterminal is present in the sentential form after walking through the cycle, meaning that the derivation does not consume more nonterminals than it produces. Therefore this derivation cannot end, i.e., there is no word generated by a walk which is longer than the length of the cycle. Now assume that the derivation terminates before starting the second run through. Then we have applied a fixed finite sequence of rules, i.e., we have generated a set L' of words such that $\#_a(w) = \#_a(w')$ for all $a \in T$ and all $w, w' \in L'$. Therefore L' is a finite set. Since we can start the derivation in any vertex whose core rule has S on its left-hand side, we can have a finite set of terminating derivations, i.e., the language generated by this programmed grammar is finite.

ii) Let $L = \{w_1, w_2, \ldots, w_n\}$ be a finite language over some alphabet T. Then we consider the programmed grammar

$$\begin{aligned} G \;=\; & (\{S\}, T, S, \{q_i \mid 1 \le i \le n\}, \{S \to w_i \mid 1 \le i \le n\}, \\ & \{(q_i, S \to w_i, \{q_{i+1}\}) \mid 1 \le i < n\} \cup \{(q_n, S \to w_n, \{q_1\})\}) \,. \end{aligned}$$

Obviously $h(G)$ is a cycle and $L(G) = L$. □

Using analogous construction we can prove the following statement.

Lemma 3. i) *Given a programmed grammar whose corresponding graph is a line, then its generated language is finite.*

ii) *Any finite language can be generated by a programmed grammar whose corresponding graph is a line.* □

Now we take into consideration the graphs, where each vertex has degree at most three.

Theorem 6. *Every language generated by a programmed grammar can be obtained by a programmed grammar having a graph where all vertices have degree at most three.*

Proof. Let $G = (N, T, S, Lab, P, P_G)$ be a programmed grammar with a graph $h(G)$ such that $h(G)$ contains a vertex α of degree $n > 3$. Assume that the in-degree of α is k and the out-degree of α is m, i.e., the situation of Figure 3 holds. Furthermore, let $A \to v$ be the rule associated with α.

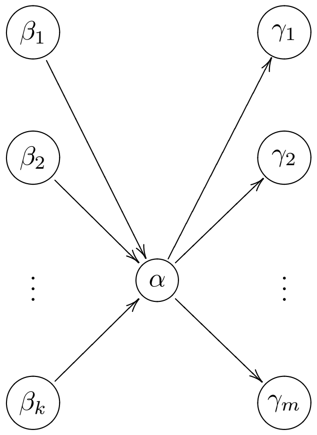

FIGURE 3. Node α with in-degree k and out-degree m

Then we substitute the subgraph of $h(G)$ shown in Figure 3 by the graph H_α presented in Figure 4 which results in a graph H_1. Since all vertices of H_α have a degree at most 3, the number of vertices with degree > 3 decreases by 1 when going from H to H_1. Furthermore, it is easy to see that $L(G) = L(g(H_1, S))$. Our construction works if $k \geq 1$ and $m \geq 2$ and m is even. The modifications for the other cases ($k = 0$ or $m = 0$ or m is odd) are left to the reader.

If H_1 contains a vertex β of degree > 3, we repeat the above construction with respect to β etc. until we obtain a graph H' where all vertices have degree at most 3. The corresponding grammar $g(H', S)$ satisfies $L(G) = L(g(H', S))$. □

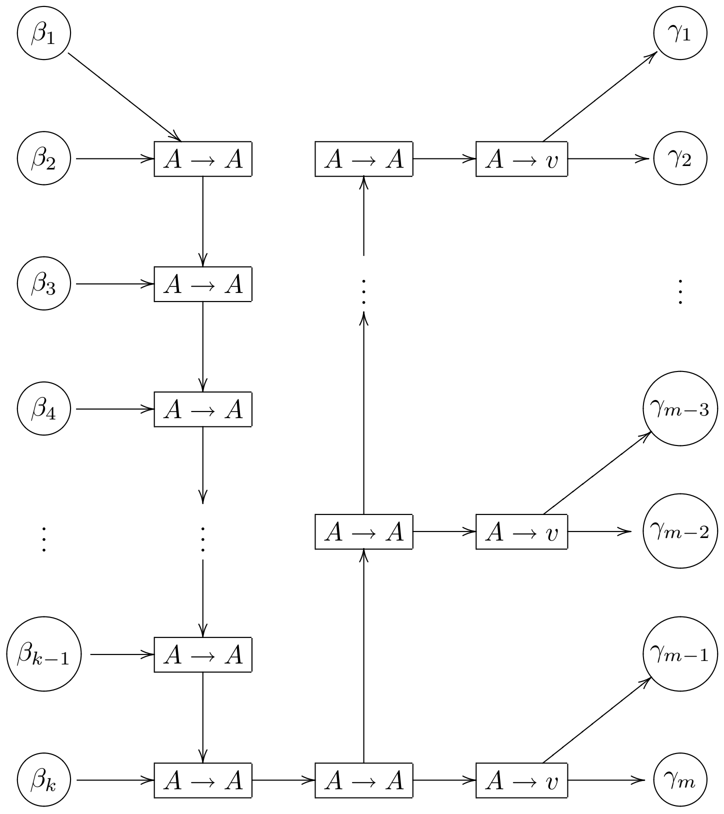

FIGURE 4. Substitution graph where all nodes have a degree at most 3

It is easy to see that for any $n \geq 3$ we have the following consequence: *any language generated by a programmed grammar can be generated by a grammar for which the graph has no vertex of degree greater than n.*

We now consider regular graphs of degree n. If $n = 2$, then the graph has to be a cycle, and Lemma 2 gives the generative power of regular graphs of degree 2.

Theorem 7. *Every language generated by a programmed grammar can be obtained by a programmed grammar with a regular graph of degree 3.*

Proof. Let L be a grammar which can be generated by a programmed grammar. By Theorem 6 we can assume that $L = L(G)$ for some programmed grammar G where any vertex of the graph $h(G)$ associated with G has a degree ≤ 3.

Let us assume that there is a vertex $A \to w$ of degree 2 in $h(G)$, i.e., we have the situation

a) $\begin{tikzcd} {} \arrow[r] & \boxed{A \to w} \arrow[r] & {} \end{tikzcd}$, *b)* $\begin{tikzcd} {} \arrow[r] & \boxed{A \to w} & {} \arrow[l] \end{tikzcd}$ or *c)* $\begin{tikzcd} {} & \boxed{A \to w} \arrow[l] \arrow[r] & {} \end{tikzcd}$.

We discuss only the situation a); for the other two cases b) and c) one can give analogous constructions. We substitute this subgraph by the graph H' given in Figure 5 where A_1 and A_2 are additional nonterminals. Let H_1 be the graph obtained by this substitution. It is easy to see that $L(G) = L(g(H_1, S))$. Moreover, going from $h(G)$ to H_1 we decrease the number of vertices with a degree < 3 by 1.

$$\begin{tikzcd}
{} & {} & \boxed{A \to A_1} \arrow[r] \arrow[ddr] & \boxed{A_1 \to A} \arrow[dr] & {} & {} \\
{} \arrow[r] & \boxed{A \to A} \arrow[ur] \arrow[dr] & {} & {} & \boxed{A \to w} \arrow[r] & {} \\
{} & {} & \boxed{A \to A_2} \arrow[r] \arrow[uur] & \boxed{A_2 \to A} \arrow[ur] & {} & {}
\end{tikzcd}$$

FIGURE 5. Substitution graph H'

Let us now assume that there is a vertex $A \to w$ of degree 1 in $h(G)$, i.e., we have the situation

d) $\begin{tikzcd} {} \arrow[r] & \boxed{A \to w} \end{tikzcd}$ or *e)* $\begin{tikzcd} {} & \boxed{A \to w} \arrow[l] \end{tikzcd}$

We only discuss case d); case e) can be handled analogously. We substitute the subgraph by the graph H'' given in Figure 6, where A_1 and A_2 are additional nonterminals, again. Let H_1' be the graph obtained by this substitution. Obviously, $L(G) = L(g(H_1', S))$. Furthermore, going from H to H_1' we decrease the number of vertices with a degree < 3 by 1.

$$\begin{tikzcd}
{} & {} & \boxed{A \to A_1} \arrow[r] \arrow[ddr] & \boxed{A_1 \to w} \arrow[dd] \\
{} \arrow[r] & \boxed{A \to A} \arrow[ur] \arrow[dr] & {} & {} \\
{} & {} & \boxed{A \to A_2} \arrow[r] \arrow[uur] & \boxed{A_2 \to w}
\end{tikzcd}$$

FIGURE 6. Substitution graph H''

By repeated applications of the above methods we finally get a graph H_2 such that $L(G) = L(g(H_2, S))$ and H_2 is a regular graph of degree 3. $\square$

Theorem 8. *Every language generated by a programmed grammar can be obtained by a programmed grammar with a regular graph of degree $n \geq 3$.*

Proof. Let G be an arbitrary programmed grammar. Then there is a programmed grammar G' such that $L(G') = L(G)$ and $h(G)$ is a regular graph of degree 3. If $n > 3$, we add to each vertex some additional vertices and edges analogous to the construction in the proof of Theorem 7 such that a regular graph of degree n is obtained. The details are left to the reader. $\square$

5. Some non-pr-universal classes of graphs

Again we start with an easy statement which is already implicitly contained in some papers/books. Since in a context-free grammar any rule can be followed by any production we have the following statement.

Lemma 4. *The family of languages generated by programmed grammars with complete graphs coincides with the family of context-free languages.* $\square$

Theorem 9. *The family of languages generated by programmed grammars with backbone graphs is incomparable with the families of regular and context-free languages (and thus it is properly included in the family of languages generated by programmed grammars).*

Proof. The graph associated with the programmed grammar G of Example 1 is a backbone graph. Its generated language $\{a^n b^n c^n \mid n \geq 2\}$ is not context-free and thus not regular.

We consider the regular language $L = \{a^n b^m \mid n \geq 1, m \geq 1\}$. We show that L cannot be generated by a programmed grammar with a backbone graph which proves the statement of the theorem.

On the contrary, suppose that there is a programmed grammar $G = (N, T, S, Lab, P, P_G)$ with a backbone graph $h(G)$ which generates L. After leaving the cycle in a derivation we can only perform a finite number of derivation steps. Thus there is a number r such that any sentential form with at least $r+1$ nonterminals cannot be terminated by leaving the cycle.

Obviously, there is a number s such that all derivations not running through the complete cycle consist of at most s derivation steps. Thus all derivations not running through the complete cycle generate a finite language only.

We now consider derivation running through the complete cycle (this is the only possibility to generate an infinite language). Let w be the sentential form if we enter the cycle. Now we run through the cycle and obtain a new sentential form v. If there is a nonterminal A such that $\#_A(w) < \#_A(v)$, we cannot run $r+1$ times through the cycle or after running $r+1$ times through the cycle we have at least $r+1$ occurrences of A in the obtained sentential form. In the former case there are at most finitely many terminating derivations from w; in the latter case we cannot terminate the derivation. Thus, if $\#_A(w) < \#_A(v)$

for some A, from w we only generate a finite set. If $\#_B(w) > \#_B(v)$ for some $B \in N$, we can perform at most $\#_B(w)$ runs through the cycle. Thus in this case we can generate at most a finite set, too.

Therefore assume that $\#_C(w) = \#_C(v)$ for all $C \in N$. Then we can arbitrarily often go through the cycle. Let n_0 and m_0 be the number of terminals of a and b, respectively, which are introduced by one run through the cycle. Assume that we go k times through the cycle and finish then the derivation using vertices outside the cycle. The generated word contains $\#_a(w)+kn_0+n_1$ occurrences of a and $\#_b(w)+km_0+m_1$ occurrences of b (where the terminating phase outside the cycle produces n_1 letters a and m_1 letters b). Since there are only a finite number of possibilities for w, (n_0, m_0) and (n_1, m_1) we cannot obtain any pair (n, m) as it is necessary in order to generate L. □

6. Discussion of related classes of grammars and languages

Above we have presented some results on the power of programmed grammars with erasing rules and without appearance checking where the control is done by restricted classes of graphs. We now discuss modifications of the concept of programmed grammars.

Programmed grammars without erasing rules and without appearance checking. Going through the proofs one sees that we have only added erasing rules in the proof of Theorem 4. Thus all the other statements on the power of programmed grammars given in this paper remains valid for programmed grammars without erasing rules (and without appearance checking).

Programmed grammars with appearance checking. A programmed grammar with appearance checking is a sixtuple $G = (N, T, S, Lab, P, P_G)$ where N, T, S, Lab and P are specified as in a programmed grammar (without appearance checking) and P_G is a finite set of quadruples $(q, A \to w, \sigma, \varphi)$ where $q \in Lab$, $A \in N$, $w \in (N \cup T)^*$ and σ and φ are subsets of Lab. Such a quadruple is applied to a sentential form $z \in (N \cup T)^+$ as follows: If $z = z_1 A z_2$ for some $z_1, z_2 \in (N \cup T)^*$, then we get $z' = z_1 w z_2$ and continue with a rule which label is in σ; and if there is no occurrence of A in z, then we do not change the sentential form and continue with a rule which label is in φ. The modelling of this type of control of the derivation process by a graph requires coloured edges. Let $q, A \to w, \sigma, \varphi)$ be a rule. The edges leading from q to a label belonging to σ have one colour, say green, whereas the edges leading from q to a rule belonging to φ have another colour, say red. However, for some graph-theoretical properties, we have problems with its definition since we have to take into consideration the colours, e.g., for Hamiltonian graphs it is not clear whether the walk through all vertices can use edges with different colours or not. If we transform a graph used for the control into a graph without coloured edges, i.e., we ignore the colours, then all results on the power of programmed grammars (without appearance checking) presented in this paper remain valid.

This statement also holds for programmed grammar without erasing rules and with appearance checking. The only critical case is that of planar graphs. Here we can use a modification of the proof of [5], Lemma 2.2.3 to get a programmed grammar with a planar graph (as control) which generates the language of a given programmed grammar. One has to note that this construction requires appearance checking.

Graph-controlled grammars. Graph-controlled grammars are distinguished from programmed grammars by that we give in advance sets of initial nodes and final nodes and the derivation has to start in an initial node and to stop in a final node. Therefore programmed grammars can be considered as graph-controlled grammars where the set of initial nodes consists of all nodes $S \to w$ (where S is the axiom) and the set of final nodes is the set of all nodes. Therefore our constructions present graph-controlled grammars where the control graphs have the required additional properties. Hence all results on the power of programmed grammars given in this paper also hold for graph-controlled grammars.

This statement also holds for graph-controlled grammars with or without appearance checking and with or without erasing rules if one takes into consideration the changes with respect to the programmed grammars given above.

We now discuss a further feature. All proofs in this paper work by the adding of nodes and edges to the given graphs. However, in some cases the added nodes, precisely the rules of the added nodes, cannot be used in terminating derivations. This holds with respect to Theorems 1, 2 and 3. It seems that corresponding statements cannot be proved by our method for graphs where all nodes are used in some terminating derivation, because we need connections (between nodes) which do not exist in the original graph and their adding leads can give new terminating derivations if all nodes can be successfully applied.

The other results on the power of programming grammars presented in the paper are valid with requirement that any rule is used in some terminating derivation, too.

Finally we mention that we have only considered some very well-known classes of graphs. However, there are some further important classes of graphs which can be investigated with respect to pr-universality.

Acknowledgement

The authors are very grateful to the referees whose critical remarks lead to a considerable improvement.

References

1. J. Dassow, *Subregularly controlled derivations: the context-free case*, Rostock. Math. Kolloq. **34** (1988), 61-70.

2. J. Dassow, *Grammars with regulated rewriting*, In: C. Martin-Vide, V. Mitrana and Gh. Paun (eds.), *Formal Languages and Applications*, 249–274, Springer-Verlag, Berlin, Heidelberg, 2004.
3. J. Dassow, *Contextual grammars with subregular choice*, Fundamenta Informaticae **64** (2005) 109–118.
4. J. Dassow and H. Hornig, *Conditional grammars with subregular conditions*, In: M. Ito and H. Jürgensen (eds.), *Proc. Internat. Conf. Words, Languages and Combinatorics II*, World Scientific, Singapore, 1994, 71-86.
5. J. Dassow and Gh. Paun, *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.
6. J. Dassow, Gh. Paun and A. Salomaa, Grammars with controlled derivations, In [13], Vol. II, 101–154.
7. R. Diestel, *Graph Theory*, Second Edition, Springer-Verlag, New York, 2000.
8. H. Fernau, *Nonterminal complexity of programmed grammars*, Theor. Comp. Sci. **296** (2003), 225–251.
9. H. Fernau, R. Freund, M. Oswald and K. Reinhardt, *Refining the nonterminal complexity of graph-controlled grammars*, In: C. Mereghetti, B. Palano, G. Pighizzini and D. Wotschke (eds.), *Proc. Descriptional Complexity of Formal Systems*, Como, 2005, 110–121.
10. J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading (Massachusetts), 1979.
11. A. Pascu and Gh. Păun, *On the planarity of bicolored digraph grammar systems*, Discr. Math. **25** (1979), 195–197
12. D. J. Rosenkrantz, *Programmed grammars and classes of formal languages*, JACM **16** (1969), 107–131.
13. G. Rozenberg and A. Salomaa, *Handbook of Formal Languages*, Vol. I – III, Springer-Verlag, Berlin, 1997.

Madalina Barbaiani, **Cristina Bibire**, **Mihai Ionescu** and **Atif Lodhi**
all Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, Spain
e-mail: {madalina.barbaiani, cristina.bibire, armandmihai.ionescu, atif.lodhi}@estudiants.urv.cat

Jürgen Dassow
Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg, Germany
e-mail: dassow@iws.cs.uni-magdeburg.de

Aidan Delaney
Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, Spain and Department of Computer Science, NUI Maynooth, Maynooth, Co. Kildare, Ireland
e-mail: adelaney@cs.may.ie

Szilárd Fazekas and **Benedek Nagy**
both Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, Spain and Faculty of Informatics, University of Debrecen, Debrecen, Hungary
e-mail: szilard.zsolt@estudiants.urv.cat , nbenedek@inf.unideb.hu

Guangwu Liu
Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, Spain and Department of Control Science and Engineering, Huazhong University of Science and Technology, Wuhan, People's Republic of China
e-mail: guanwu.liu@estudiants.urv.cat