A NEW NORMAL FORM FOR PROGRAMMED GRAMMARS

Lukáš Vrábel
Doctoral Degree Programme (1), FIT BUT
E-mail: xvrabe01@stud.fit.vutbr.cz

Supervised by: Alexander Meduna
E-mail: meduna@fit.vutbr.cz

Abstract: In the present paper, we discuss programmed grammars. We investigate the effect of the number of rules with more than one successor on generative power of the programmed grammars. We prove that for every programmed grammar, there is an equivalent programmed grammar where only a single rule has more than one successor.

Keywords: Programmed grammar, complexity, normal form, successor

1 INTRODUCTION

In the formal language theory, programmed grammars have been thoroughly investigated (see [1, 2, 4–6] for recent studies). Although various properties of these grammars have been well established, the effect of rules with more than one successor has not been investigated to its full extent. In [1] and [2], it is proved that (a) to generate an infinite language, there has to be at least one rule with more than one successor, and (b) any programmed grammar can be converted to an equivalent programmed grammar with every rule having at most two successors. However, there has been no study of maximum needed number of rules with more than one successor.

In this paper, we introduce a new normal form for programmed grammars, called the *one-ND rule normal form* (ND stands for *n*on*d*eterministic), where at most one rule has more than one successor. We prove that every programmed grammar can be converted to this form.

2 DEFINITIONS

This paper assumes that the reader is familiar with the theory of formal languages (see [7]), including the theory of regulated rewriting (see [3]). For a set, Q, $\mathrm{card}(Q)$ denotes the cardinality of Q, and 2^Q denotes the power set of Q. For an alphabet, V, V^* represents the free monoid generated by V under the operation of concatenation. The unit of V^* is denoted by ε. Set $V^+ = V^* - \{\varepsilon\}$; algebraically, V^+ is thus the free semigroup generated by V under the operation of concatenation.

Definition 1. A *programmed grammar* (see [3, 8]) is a quintuple, $G = (N, T, S, \Psi, P)$, where N is the alphabet of *nonterminals*, T is the alphabet of *terminals*, $N \cap T = \emptyset$, $S \in N$ is the *start symbol*, Ψ is the alphabet of *rule labels*, and $P \subseteq \Psi \times N \times (N \cup T)^* \times 2^\Psi$ is a finite relation such that $\mathrm{card}(\Psi) = \mathrm{card}(P)$, and for $(r,A,x,\sigma_r), (q,B,y,\sigma_q) \in P$, if $(r,A,x,\sigma_r) \neq (q,B,y,\sigma_q)$, then $r \neq q$.

Elements of P are called *rules*. Instead of $(r,A,x,\sigma_r) \in P$, we write $\lfloor r\colon A \to x, \sigma_r \rfloor \in P$ throughout this paper. For $\lfloor r\colon A \to x, \sigma_r \rfloor \in P$, A is referred to as the *left-hand side* of r, and x is referred to as the *right-hand side* of r. Let $V = N \cup T$ be the *total alphabet*. G is *propagating* if every $\lfloor r\colon A \to x, \sigma_r \rfloor \in P$ satisfies $x \in V^+$. Rules of the form $\lfloor r\colon A \to \varepsilon, \sigma_r \rfloor$ are called *erasing rules*.

The relation of a *direct derivation*, symbolically denoted by $\Rightarrow$, is defined over $V^* \times \Psi$ as follows: for $(x_1, r), (x_2, s) \in V^* \times \Psi$, $(x_1, r) \Rightarrow (x_2, s)$ (or $(x_1, r) \Rightarrow_G (x_2, s)$, if there is a danger of confusion) if and only if $x_1 = yAz$, $x_2 = ywz$, $\lfloor r\colon A \to w, \sigma_r \rfloor \in P$, and $s \in \sigma_r$.

Let $\lfloor r\colon A \to w, \sigma_r \rfloor \in P$. Then, σ_r is called the *success field* of r. Let $\Rightarrow^n$, $\Rightarrow^*$, and $\Rightarrow^+$ denote the nth power of $\Rightarrow$, for some $n \geq 0$, the reflexive-transitive closure of $\Rightarrow$, and the transitive closure of $\Rightarrow$, respectively. Let $(S,r) \Rightarrow^* (w,s)$, where $r,s \in \Psi$ and $w \in V^*$. Then, (w,s) is called a *configuration*. The *language generated by* G is denoted by $L(G)$ and defined as $L(G) = \{w \in T^* \mid (S,r) \Rightarrow^* (w,s)$, for some $r,s \in \Psi\}$. ■

Definition 2. Let $G = (N, T, S, \Psi, P)$ be a programmed grammar. G is in the *one-ND rule normal form* (ND stands for *n*on*d*eterministic) if at most one $\lfloor r\colon A \to x, \sigma_r \rfloor \in P$ satisfies $\text{card}(\sigma_r) \geq 1$ and every other $\lfloor r\colon A \to x, \sigma_r \rfloor \in P$ satisfies $\text{card}(\sigma_r) \leq 1$. ■

3 RESULTS

The following algorithm converts any programmed grammar, G, to an equivalent programmed grammar in the one-ND rule normal form, G'. To give an insight into this conversion, we first explain the underlying idea behind it. First, we introduce the only nondeterministic rule of G', $\lfloor X\colon \# \to \#, \sigma_X \rfloor$. Obviously, each nondeterministic choice of some rule $\lfloor r\colon A \to x, \{s_1, s_2, \ldots, s_n\} \rfloor$ of G has to be simulated using X. To ensure proper simulation, we have to satisfy that (i) one of s_i is applied after r, and (ii) no other rules can be applied after r.

To satisfy both of the requirements, we introduce a special nonterminal symbol, $\langle r \rangle$, for each rule r of G. These symbols are used to store the information about the last applied rule in a derivation. Then, for each successor of r, s_i, we introduce the following sequence of rules:

- $\lfloor r\colon A \to x, \{r_\sigma\} \rfloor$ to simulate r,
- $\lfloor r_\sigma\colon \langle \emptyset \rangle \to \langle r \rangle, \{X\} \rfloor$ to preserve the information that r was the last applied rule,
- X to make a nondeterministic choice of the successor of r, and
- $\lfloor \langle r \rhd s_i \rangle\colon \langle r \rangle \to \langle \emptyset \rangle, \{s_i\} \rfloor$ to check if r was the last rule of G applied in a derivation before this rule.

Note that if X chooses some $\langle p \rhd q \rangle$ with $p \neq r$ instead, the derivation gets blocked because $\langle p \rangle$ is not present in the current sentential form. Finally, to derive a string of terminals, we have to introduce a nondeterministic choice to erase $\langle r \rangle$ and #. This is done by rules introduced in (3) in the algorithm.

Algorithm 1. Conversion of any programmed grammar to the one-ND rule normal form.

Input: A programmed grammar $G = (N, T, S, \Psi, P)$.

Output: A programmed grammar in the one-ND rule normal form, $G' = (N', T, S', \Psi', P')$, such that $L(G') = L(G)$.

Method: Initially, set:

$N' = N \cup \{\#, \langle \emptyset \rangle, S'\} \cup \{\langle r_\sigma \rangle \mid r \in \Psi\}$;
$\Psi' = \Psi \cup \{X\}$ with X being a new unique symbol;
$P' = \{\lfloor r\colon A \to x, \sigma_r \rfloor \mid \lfloor r\colon A \to x, \sigma_r \rfloor \in P, \text{card}(\sigma_r) = 1\} \cup \{\lfloor X\colon \# \to \#, \sigma_X \rfloor\}$ with σ_X initially set to $\emptyset$.

Now, apply the following three steps:

(1) for each $\lfloor r\colon A \to \omega, \sigma_r \rfloor \in P$ satisfying $\text{card}(\sigma_r) > 1$:

(1.1) add $\lfloor r\colon A \to \omega, \{r_\sigma\} \rfloor$ to P',
(1.2) add $\lfloor r_\sigma\colon \langle\emptyset\rangle \to \langle r_\sigma\rangle, \{X\} \rfloor$ to P', and r_σ to Ψ',
(1.3) for each $q \in \sigma_r$, add $\lfloor \langle r \rhd q\rangle\colon \langle r_\sigma\rangle \to \langle\emptyset\rangle, \{q\} \rfloor$ to P', $\langle r \rhd q\rangle$ to Ψ', and $\langle r \rhd q\rangle$ to σ_X;
(2) for each $\lfloor r\colon S \to \omega, \sigma_r \rfloor \in P'$:
(2.1) add $\lfloor r_s\colon S' \to \#\langle\emptyset\rangle S, \{r\} \rfloor$ to P',
(2.2) add $\lfloor r_{sf}\colon S' \to S, \{r\} \rfloor$ to P',
(2.3) add r_s and r_{sf} to Ψ';
(3) for each $\lfloor \langle p \rhd q\rangle\colon \langle p\rangle \to \langle\emptyset\rangle, \{q\} \rfloor \in P'$ satisfying $\langle p \rhd q\rangle \in \sigma_X$:
(3.1) add $\lfloor \langle p \rhd q, f_1\rangle\colon \langle p\rangle \to \varepsilon, \{\langle p \rhd q, f_2\rangle\} \rfloor$ to P',
(3.2) add $\lfloor \langle p \rhd q, f_2\rangle\colon \# \to \varepsilon, \{q\} \rfloor$ to P',
(3.3) add $\langle p \rhd q, f_1\rangle$ to σ_X,
(3.4) add $\langle p \rhd q, f_1\rangle$ and $\langle p \rhd q, f_2\rangle$ to Ψ'; ■

Lemma 1. *Algorithm 1 is correct.*

Proof. Clearly, the algorithm always halts and G' is in the one-ND rule normal form. To establish $L(G) = L(G')$, we first prove $L(G) \subseteq L(G')$ by showing how derivations of G are simulated by G', and then we prove $L(G') \subseteq L(G)$ by showing how every $s \in L(G')$ can be generated by G.

Set $V = N \cup T$ and $\bar{N} = N' - N$. Observe that all strings derived from S' in G' are of the form $\#\langle z\rangle u$, $\#u$, or u, where $\langle z\rangle \in \bar{N}$, $u \in V^*$.

Due to the size constraint of this paper, the proofs of the following three claims are left to the reader.

Claim 1. *If* $(u, r) \Rightarrow_G (w, q)$, *then* $(\#\langle\emptyset\rangle u, r) \Rightarrow^*_{G'} (\#\langle\emptyset\rangle w, q)$, *where* $u, w \in V^*$, $r, q \in \Psi$.

Claim 2. *If* $(S', \alpha) \Rightarrow^*_{G'} (\#\langle z\rangle w, q)$, *where* $\alpha, q \in \Psi', w \in V^*$, *and* $\langle z\rangle \in \bar{N}$, *then* $(S', \alpha') \Rightarrow^*_{G'} (w, q')$, *where* $\alpha', q' \in \Psi'$.

Claim 3. *If* $(S', \alpha') \Rightarrow^*_{G'} (w, q')$, *then* $(S', \alpha) \Rightarrow^*_{G'} (\#\langle\emptyset\rangle w, q)$, *where* α, α', $q' \in \Psi'$, $q \in \Psi$, *and* $w \in V^*$.

Claim 1 establishes the relation between the derivation step in G and its counterpart in G'. Claims 2 and 3 show the relation between $w \in V^*$ derived in G' from S', and its corresponding sentence form, $\#\langle z\rangle w$, containing the symbol used to preserve the information about the last rule applied in a derivation.

Following claim demonstrates how derivations of G are simulated by G'.

Claim 4. *Let* $(S, r) \Rightarrow^m_G (w, q)$, *where* $r, q \in \Psi$, $w \in V^*$, *for some* $m \geq 1$. *Then,* $(S', r_s) \Rightarrow^*_{G'} (\#\langle\emptyset\rangle w, q)$, *where* $r_s \in \Psi'$.

Proof. This claim is established by induction on m, $m \geq 1$.

Basis. Let $m = 1$. Then, $(S, r) \Rightarrow_G (w, q)$ by some $r \in \Psi$. By Claim 1, $(\#\langle\emptyset\rangle S, r) \Rightarrow^*_{G'} (\#\langle\emptyset\rangle w, q)$. Since r has S on its left-hand side, $\lfloor r_s\colon S' \to \#\langle\emptyset\rangle S, \{r\} \rfloor \in P'$ by (2.1), so $(S', r_s) \Rightarrow_{G'} (\#\langle\emptyset\rangle S, r) \Rightarrow^*_{G'} (\#\langle\emptyset\rangle w, q)$. Thus, the basis holds.

Induction Hypothesis. Suppose that the claim holds for all derivations of length l or less, where $l \leq m$, for some $m \geq 1$.

Induction Step. Consider any derivation of the form $(S, r) \Rightarrow^{m+1}_G (w, q)$, where $w \in V^*$ and $r, q \in \Psi$. Since $m + 1 \geq 1$, this derivation can be expressed as $(S, r) \Rightarrow^m_G (x, p) \Rightarrow_G (w, q)$, where $x \in V^*$, $p \in \Psi$. By the induction hypothesis, $(S', r_s) \Rightarrow^*_{G'} (\#\langle\emptyset\rangle x, p)$, and by Claim 1, $(\#\langle\emptyset\rangle x, p) \Rightarrow^*_{G'} (\#\langle\emptyset\rangle w, q)$. Thus, the claim holds. □

Now, we show that for each derivation of $\#\langle\emptyset\rangle u$ in G', there is a derivation of u in G, which will be later used to prove $L(G') \subseteq L(G)$.

Claim 5. *If* $(S', r_s) \Rightarrow_{G'}^{m} (\#\langle\emptyset\rangle u, q)$, *for some* $m \geq 1$, *then* $(S, r) \Rightarrow_{G}^{*} (u, q)$, *where* $r, q \in \Psi$, $r_s \in \Psi'$, *and* $u \in V^*$.

Proof. This claim is established by induction on m, $m \geq 1$.

Basis. Let $m = 1$. Then, $(S', r_s) \Rightarrow_{G'} (\#\langle\emptyset\rangle S, r)$. As r_s is created in (2.1) from $r \in \Psi$, $(S, r) \Rightarrow_{G}^{0} (S, r)$, so the basis holds.

Induction Hypothesis. Suppose that the claim holds for all derivations of length l or less, where $l \leq m$, for some $m \geq 1$.

Induction Step. Consider any $(S', r_s) \Rightarrow_{G'}^{m+1} (\#\langle\emptyset\rangle u, q)$, where $u \in V^*$ and $r_s, q \in \Psi$. Since $m + 1 \geq 2$, this derivation can be expressed as

$$(S', r_s) \Rightarrow_{G'}^{m} (\#\langle z\rangle v, p') \Rightarrow_{G'} (\#\langle\emptyset\rangle u, q),$$

where $v \in V^*$, $\langle z\rangle \in \bar{N}$, and $p' \in \Psi'$. As $\Psi \subseteq \Psi'$, there are two cases, (i) and (ii), based on whether $p' \in \Psi$ or $p' \in \Psi' - \Psi$:

(i) Assume that $p' \in \Psi$. As $q \in \Psi$, $\lfloor p' \colon A \to x, \{q\} \rfloor \in P'$ is one of the rules created in the initialization part of the algorithm, so it is also in P. Therefore, $(v, p') \Rightarrow_{G} (u, q)$ and $\langle z\rangle = \langle\emptyset\rangle$. By the induction hypothesis, $(S, r) \Rightarrow_{G}^{*} (v, p')$, so $(S, r) \Rightarrow_{G}^{*} (u, q)$. Thus, the claim holds for this case.

(ii) Assume that $p' \in \Psi' - \Psi$. As $q \in \Psi$, p' has to be one of the $\langle p \rhd q\rangle$ created in (1.3) from some $\lfloor p \colon A \to x, \sigma_p \rfloor \in P$ (observe that only rules created in (1.3) have their label in $\Psi' - \Psi$ and have some $q \in \Psi$ in their success field). As $\lfloor \langle p \rhd q\rangle \colon \langle p\rangle \to \langle\emptyset\rangle, \{q\} \rfloor$ has $\langle p\rangle$ on its left-hand side, $\langle z\rangle = \langle p\rangle$ and $v = u$. Since the only success field containing $\langle p \rhd q\rangle$ is σ_X, the derivation has to be of the form

$$(S', r_s) \Rightarrow_{G'}^{m-1} (\#\langle p\rangle u, X) \Rightarrow_{G'} (\#\langle p\rangle u, \langle p \rhd q\rangle) \Rightarrow_{G'} (\#\langle\emptyset\rangle u, q).$$

Note that X is only in the success field of rules created in (1.2). Furthermore, note that $\langle p\rangle$ can be generated only rules created in (1.2). So, there has to be $\lfloor p_\sigma \colon \langle\emptyset\rangle \to \langle p\rangle, \{X\} \rfloor \in P'$, created in (1.2), such that the derivation can be expressed as

$$(S', r_s) \Rightarrow_{G'}^{m-2} (\#\langle\emptyset\rangle u, p_\sigma) \Rightarrow_{G'} (\#\langle p\rangle u, X) \Rightarrow_{G'}^{2} (\#\langle\emptyset\rangle u, q).$$

As p_σ was created from $\lfloor p \colon A \to x, \sigma_p \rfloor \in P$, where $A \in N$ and $x \in V^*$, there is corresponding $\lfloor p \colon A \to x, \{p_\sigma\} \rfloor \in P'$, and it is the only rule containing p_σ in its success field. Therefore, the derivation can be expressed as

$$(S', r_s) \Rightarrow_{G'}^{m-3} (\#\langle\emptyset\rangle v', p) \Rightarrow_{G'} (\#\langle\emptyset\rangle u, p_\sigma) \Rightarrow_{G'}^{3} (\#\langle\emptyset\rangle u, q),$$

where $v' \in V^*$. Since $\lfloor p \colon A \to x, \sigma_p \rfloor \in P$, $(v', p) \Rightarrow_{G} (u, q)$. By the induction hypothesis, $(S, r) \Rightarrow_{G}^{*} (v', p)$, so $(S, r) \Rightarrow_{G}^{*} (u, q)$. Thus, the claim holds for this case.

Observe, that these two cases cover all possible forms of $(S', r_s) \Rightarrow_{G'}^{*} (\#\langle\emptyset\rangle u, q)$. Thus, the claim holds. □

To establish $L(G) = L(G')$, it suffices to show the following two statements:

- by Claim 4, for each $(S,r) \Rightarrow_G^* (u,q)$, where $r,q \in \Psi$, and $u \in T^*$, there is $(S',r_s) \Rightarrow_{G'}^* (\#\langle\emptyset\rangle u,q)$, where $r_s \in \Psi'$. Then, $(S',r') \Rightarrow_{G'}^* (u,q')$ by Claim 2, so $L(G) \subseteq L(G')$.
- by Claim 3, for each $(S',r') \Rightarrow_{G'}^* (u,q')$, where $r',q' \in \Psi'$ and $u \in T^*$, there is $(S',r_s) \Rightarrow_{G'}^* (\#\langle\emptyset\rangle u,q)$, where $r_s \in \Psi'$ and $q \in \Psi$. Then, $(S,r) \Rightarrow_G^* (u,q)$, where $r \in \Psi$, by Claim 5, so $L(G') \subseteq L(G)$.

As $L(G) \subseteq L(G')$ and $L(G') \subseteq L(G)$, $L(G) = L(G')$, so the lemma holds. □

The following theorem represents the main achievement of this paper.

Theorem 1. *For any programmed grammar, G, there is a programmed grammar in the one-ND rule normal form, G′, such that* $L(G') = L(G)$.

Proof. This theorem follows from Algorithm 1 and Lemma 1. □

4 CONCLUSION

In this section, we present some open problems. First, consider *programmed grammars with appearance checking* (see [3]). Throughout this paper, we have investigated only programmed grammars without appearance checking. Does the achieved result also hold in terms of programmed grammars with appearance checking?

Second, observe that Algorithm 1 introduces erasing rules to G', even if the input grammar is propagating. Can the algorithm be modified in such way, that when G is propagating, then so is G'?

REFERENCES

[1] M. Barbaiani, C. Bibire, J. Dassow, A. Delaney, S. Fazekas, M. Ionescu, G. Liu, A. Lodhi, and B. Nagy. The power of programmed grammars with graphs from various classes. *Journal of Applied Mathematics & Computing*, 22(1–2):21–38, 2006.

[2] H. Bordihn and M. Holzer. Programmed grammars and their relation to the LBA problem. *Acta Informatica*, 43(4):223–242, 2006.

[3] J. Dassow and G. Păun. *Regulated Rewriting in Formal Language Theory*. Springer, 1989.

[4] H. Fernau. Nonterminal complexity of programmed grammars. *Theoretical Computer Science*, 296(2):225–251, 2003.

[5] H. Fernau, R. Freund, M. Oswald, and K. Reinhardt. Refining the nonterminal complexity of graph-controlled, programmed, and matrix grammars. *Journal of Automata, Languages and Combinatorics*, 12(1–2):117–138, 2007.

[6] H. Fernau and F. Stephan. Characterizations of recursively enumerable sets by programmed grammars with unconditional transfer. *Journal of Automata, Languages and Combinatorics*, 4(2):117–152, 1999.

[7] A. Meduna. *Automata and Languages: Theory and Applications*. Springer, London, 2000.

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