Weak Greibach Normal Form for Hyperedge Replacement Grammars

Tikhon Pshenitsyn

Department of Mathematical Logic and Theory of Algorithms [*]
Faculty of Mathematics and Mechanics
Lomonosov Moscow State University
GSP-1, Leninskie Gory, Moscow, 119991, Russian Federation

`tpshenitsyn@lpcs.math.msu.su`

It is known that hyperedge replacement grammars are similar to string context-free grammars in the sense of definitions and properties. Therefore, we expect that there is a generalization of the well-known Greibach normal form from string grammars to hypergraph grammars. Such generalized normal forms are presented in several papers; however, they do not cover a large class of hypergraph languages (e.g. languages consisting of star graphs). In this paper, we introduce a weak Greibach normal form, whose definition corresponds to the lexicalized normal form for string grammars, and prove that every context-free hypergraph language (with nonsubstantial exceptions) can be generated by a grammar in this normal form. The proof presented in this paper generalizes a corresponding one for string grammars with a few more technicalities.

1 Introduction

Extensions of formal grammars and languages from strings to graphs are considered in a wide variety of works. The resulting formalisms called graph grammars generate graph languages by means of productions: each production is of the form $A \to G$; it allows one to replace a part of a graph labeled by A with the graph G if certain conditions are satisfied. An overview on graph grammars can be found in "Handbook on Graph Grammars and Computing by Graph Transformation" [11].

In this paper, we focus on a particular approach called hyperedge replacement grammar (HRG). An overview on HRG can be found in the book [7] or in a chapter of the handbook mentioned above [3]. In brief, a hyperedge replacement grammar contains productions that allow one to replace an edge with a certain label by a hypergraph; the rest is similar to string context-free grammar definitions. HRGs appeared in the seventies, and they became popular from theoretical and practical points of view; particularly, they can be used in machine translation or programming (see, for instance, [9]). Many structural properties of HRGs have been studied. Nicely, many theorems regarding context-free string grammars (CFGs) can be extended to HRGs in a natural way; besides, it is often the case that proofs for CFGs can be generalized to HRGs so there is no need to invent anything new.

For context-free grammars, there is a so-called Greibach normal form, which is helpful in a number of investigations, e.g. to connect CFGs with categorial grammars (see [2]). Each production in a grammar in the Greibach normal form has to begin with a terminal symbol and proceed with at most two nonterminal ones. This definition can be weakened: one says that a grammar is in the weak Greibach normal form (it is often called *lexicalized*) if there is exactly one terminal symbol in the right-hand side of each production. Obviously, the same definition can be introduced for graphs. Namely, an HRG is said to be

---

in the weak Greibach normal form if for each production $A \to H$ the hypergraph H has exactly one edge labeled by a terminal symbol.

The main objective of this paper is to present a precise class of graph languages that can be generated by HRGs in the weak Greibach normal form. In order to do this we generalize the proof of existence of the Greibach normal form for context-free grammars taken from [1] in a straightforward way. However, several differences exist. Firstly, there is a nonsubstantial class of hypergraph context-free languages that cannot be generated by grammars in the weak Greibach normal form (a problem arises because of isolated nodes). Secondly, there is a trick in the proof for CFGs that exploits the string nature of grammars, so it is hard to generalize it to hypergraphs. This led us to a large and technically heavy proof. Finally, there are bad news regarding algorithmical complexity: conversion of an HRG into an equivalent one in the weak Greibach normal form cannot be done even in exponential time.

In Sect. 2 we compare our contribution with other studies related to the Greibach normal form for graph grammars. In Sect. 3 formal definitions related to hypergraphs and HRGs are presented. In Sect. 4 the weak Greibach normal form is introduced, related issues are discussed. Sect. 5 is devoted to the main theorem. In Sect. 6 we discuss algorithmic complexity of the normalization procedure. In Sect. 7 we conclude.

2   Related Work

There are several works devoted to the Greibach normal form for graph grammars. The paper of Joost Engelfriet [4] establishes that HRGs that produce *languages of bounded degree* are equivalent to *apex HRGs*; this result generalizes *the double Greibach normal form* for CFGs. To recall, a language L is of bounded degree if for some M all nodes of all hypergraphs in L have degree not exceeding M. Obviously, there are substantive examples of languages of unbounded degree: for instance, the language of star graphs, which have one node and arbitrarily many edges outgoing from it. Besides, the property of being apex is stronger than the weak Greibach normal form we are interested in.

In the paper of Christina Jansen et al. [8] *the local Greibach normal form* is presented. The authors prove that *data structure grammars* (it is a specific kind of grammars that generate so-called heap configurations) can be transformed into grammars in the local Greibach normal form; after the proof the authors point out that the normalization can be generalized to HRGs of bounded degree. However, the authors also note that their procedure being algorithmically efficient cannot be generalized to all HRGs.

The weak Greibach normal form in the sense we are interested in is introduced for another type of graph grammars. Namely, in [5] it is proved that each B-eNCE grammar (here we do not consider definitions regarding this formalism) is equivalent to a B-eNCE grammar in the weak Greibach normal form. In [6] it is shown that B-edNCE grammars and HRGs have the same recognizing power in some sense (namely, since B-edNCE grammars produce usual graphs with labeled nodes and edges while HRGs produce hypergraphs with labeled edges, the result is established w.r.t. two translations, from graphs to hypergraphs and vice versa). However, it seems to be impossible that these two results can be combined in order to obtain the normal form for HRGs.

Therefore, to our best knowledge, the question of whether each HRG is equivalent to an HRG in the weak Greibach normal form has hitherto remained open. We answer it in this paper.

3   Preliminaries

This section is concerned with definitions related to hypergraphs. All of them are taken from [3]. Note that we use a slightly different notation from that in the handbook mentioned above.

3.1   Hypergraphs, Sub-hypergraphs

$\mathbb{N}$ includes 0. The set of integers from 1 to n is denoted by $[1,n]$.

It is convenient for us to use the following notation: if $\{i_1,\ldots,i_k\}$ is an indexed set of integers such that for $m < n$ $i_m < i_n$ holds, then it is called index-ordered and the set is denoted as $\{i_1,\ldots,i_k\}_{IO}$. The set Σ^* is the set of all strings over the alphabet Σ including the empty string ε. The length $|w|$ of the word w is the number of positions in w. The i-th symbol in w is denote by $w(i)$ ($1 \le i \le |w|$). Σ^+ denotes the set of all nonempty strings. The set $\Sigma^\circledast$ is the set of all strings consisting of distinct symbols. The set of all symbols contained in the word w is denoted by $[w]$. If $f : \Sigma \to \Delta$ is a function from one set to another, then it is naturally extended as a function $f : \Sigma^* \to \Delta^*$ ($f(\sigma_1 \ldots \sigma_k) = f(\sigma_1) \ldots f(\sigma_k)$).

Let C be some fixed set of labels, for which the function $type : C \to \mathbb{N}$ is considered.

Definition 1. *A hypergraph over C is a tuple $G = \langle V, E, att, lab, ext \rangle$ where V is the set of nodes, E is the set of hyperedges, $att : E \to V^\circledast$ assigns an ordered set of attachment nodes to each edge, $lab : E \to C$ labels each hyperedge by some element of C in such a way that $type(lab(e)) = |att(e)|$ whenever $e \in E$, and $ext \in V^\circledast$ is an ordered set of external nodes.*

Components of a hypergraph G are denoted by $V_G, E_G, att_G, lab_G, ext_G$.

In the remainder of the paper, hypergraphs are simply called graphs, and hyperedges are simply called edges. The set of all graphs with labels from C is denoted by $\mathscr{H}(C)$. In this work, figures contain graph drawings and sketches. In drawings, nodes are depicted by black dots, edges are denotes as labeled boxes, att is represented with numbered lines (called "tentacles"), external nodes are depicted by numbers in brackets. If an edge has type 2, it is depicted by an arrow. If we are not interested in a specific form of a graph but we want to have a closer look at some its part, we depict the whole graph as an area but draw its part of interest in detail.

Example 1. This is a graph:
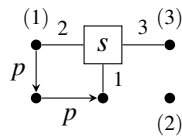
If G is a graph, and $e \in E_G$ is labeled by a, then G can be denoted by $G(e : a)$.

Definition 2. The function $type$ (or $type_G$ to be exact) returns the number of nodes attached to some edge in a graph G: $type_G(e) := |att_G(e)|$. If G is a graph, then $type(G) := |ext_G|$.

Definition 3. A sub-hypergraph (or just subgraph) H of a hypergraph G is a hypergraph such that $V_H \subseteq V_G$, $E_H \subseteq E_G$, and for all $e \in E_H$ $att_H(e) = att_G(e)$, $lab_H(e) = lab_G(e)$.

Definition 4. If $H = \langle \{v_i\}_{i=1}^n, \{e_0\}, att, lab, v_1 \ldots v_n \rangle$, $att(e_0) = v_1 \ldots v_n$ and $lab(e_0) = a$, then H is called *a handle*. It is denoted by $\odot(a)$.

Definition 5. *An isomorphism* between graphs G and H is a pair of bijective functions $\mathscr{E} : E_G \to E_H$, $\mathscr{V} : V_G \to V_H$ such that $att_H \circ \mathscr{E} = \mathscr{V} \circ att_G$, $lab_G = lab_H \circ \mathscr{E}$, $\mathscr{V}(ext_G) = ext_H$. In this work, we do not distinguish between isomorphic graphs.

3.2 Replacement

This procedure is defined in [3]. In short, the replacement of an edge e_0 in G with a graph H can be done if $type(e_0) = type(H)$ as follows:

1. Remove e_0;

2. Insert an isomorphic copy of H (namely, H and G have to consist of disjoint sets of nodes and edges);

3. For each i, fuse the i-th external node of H with the i-th attachement node of e_0.

To be more precise, the set of edges in the resulting graph is $(E_G \setminus \{e_0\}) \cup E_H$, and the set of nodes is $V_G \cup (V_H \setminus ext_H)$. The result is denoted by $G[H/e_0]$.

3.3 Hyperedge Replacement Grammars

Definition 6. A *hyperedge replacement grammar (HRG)* is a tuple $HGr = \langle N, \Sigma, P, S \rangle$, where N is a finite alphabet of nonterminal symbols, Σ is a finite alphabet of terminal symbols ($N \cap \Sigma = \emptyset$), P is a set of productions, and $S \in N$. Each production is of the form $A \to H$ where $A \in N$, $H \in \mathcal{H}(N \cup \Sigma)$ and $type(A) = type(H)$. For $\pi = A \to H$ we denote $lhs(\pi) = A, rhs(\pi) = H$.

Edges labeled by terminal (nonterminal) symbols are called *terminal edges* (*nonterminal edges* resp.). If a graph contains terminal edges only, it is called *terminal*.

If G is a graph, $e_0 \in E_G$, $lab(e_0) = A$ and $\pi = A \to H \in P$, then G directly derives $G[H/e_0]$ (we write $G \Rightarrow G[H/e_0]$ or $G \underset{\pi}{\Rightarrow} G[H/e_0]$). The transitive reflexive closure of $\Rightarrow$ is denoted by $\overset{*}{\Rightarrow}$. If $G \overset{*}{\Rightarrow} H$, then G is said to derive H. The corresponding sequence of production applications is called a derivation.

Definition 7. The *language generated by an HRG HGr* $= \langle N, \Sigma, P, S \rangle$ is the set of graphs $H \in \mathcal{H}(\Sigma)$ such that $\odot(S) \overset{*}{\Rightarrow} H$. It is denoted by $L(HGr)$.

A graph language L is said to be *a hypergraph context-free language (HCFL)* if it is generated by some HRG.

Two grammars are said to be equivalent if they generate the same language.

Further we simply write $A \overset{*}{\Rightarrow} G$ instead of $\odot(A) \overset{*}{\Rightarrow} G$.

4 Weak Greibach Normal Form

In this section we introduce the formal definition of the normal form we are interested in, and establish a simple property of languages generated by grammars in this normal form.

Definition 8. An HRG HGr is in *the weak Greibach normal form (WGNF)* if there is exactly one terminal edge in the right-hand side of each production. Formally, $\forall (X \to H) \in P_{HGr} \exists! e_0 \in E_H : lab_H(e_0) \in \Sigma_{HGr}$.

Example 2. The grammar $HGr_1 = \langle \{S\}, \{a\}, P_1, S \rangle$ is in the WGNF where P_1 contains the following rules ($type(S) = type(a) = 1$):
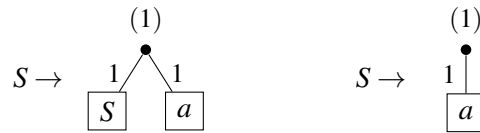
This grammar generates the language of star 1-edged a-labeled hypergraphs, which is unbounded.

Remark 1. If $\langle N, \Sigma, P, S \rangle$ is in the WGNF and $X \overset{k}{\Rightarrow} G$ for some $X \in N$, then G has exactly k terminal edges (each production adds exactly one terminal edge).

Note that not each hypergraph context-free language can be generated by some HRG in the WGNF. This follows from

Example 3. Consider the HRG $HGr_2 = \langle \{T\}, \{b\}, P_2, T \rangle$ where P_2 contains the following rules $(type(T) = type(b) = 0)$:

$$T \rightarrow \boxed{T} \qquad\qquad T \rightarrow \boxed{b}$$
$$\bullet$$

The first production just adds an isolated node; thus this grammar produces graphs that have exactly one edge labeled by b and arbitrarily many isolated nodes. If there is an equivalent $HGr' = \langle N, \{b\}, P', S' \rangle$ in the WGNF, then each right-hand side of each production in P' contains exactly one terminal edge. Note that if $S' \overset{k}{\Rightarrow} G, G \in \mathscr{H}(\{b\})$ for some k in HGr', then G has k terminal edges (Remark 1); hence k has to be equal to 1 and therefore $S' \rightarrow G \in P'$. However, there are infinitely many graphs in $L(HGr)$ while $|P'| < \infty$. □

It is seen that the reason of this problem is in isolated nodes. In the string case if a language contains the empty word, then it cannot be generated by a grammar in the WGNF due to obvious reasons: each production adds at least one terminal symbol so it is impossible to produce the empty word. In the case of graphs, we also have to prohibit somehow undesired languages with "too many" isolated nodes occuring in graphs.

Below we describe the precise class of languages generated by grammars in the weak Greibach normal form. We denote by $esize(G)$ the number of edges in G, and by $isize(G)$ the number of isolated nodes in G.

Definition 9. An HCFL L is said to be *isolated-node bounded* (ibHCFL) if there is a constant M such that, for each $G \in L$, $isize(G) < M * esize(G)$. An HRG HGr is called isolated-node bounded if it generates an ibHCFL.

Theorem 1. *Each HRG $HGr = \langle N, \Sigma, P, S \rangle$ in the WGNF generates an ibHCFL.*

Proof. Let $M = \max_{A \rightarrow H \in P} \{isize(H)\} + 1$. Define $esize_T(G)$ as the number of terminal edges in G. We prove by induction on k that for each graph G such that $X \overset{k}{\Rightarrow} G, X \in N$ $isize(G) < M * esize_T(G)$. Remark 1 implies that $esize_T(G) = k$.

Basis. If $k = 0$, then $G = \odot(X)$, and the statement is trivial ($0 < M$).

Step. Let $X \overset{k-1}{\Rightarrow} H \Rightarrow G$. By the induction hypothesis, $isize(H) < M * esize_T(H)$. The number of isolated nodes appeared at the last step does not exceed M by the definition of M. Then $isize(G) < M * esize_T(H) + M = M * esize_T(G)$.

Note finally that $esize_T(G) \leq esize(G)$. □

The other direction of this statement is of central interest in this paper. The next section is devoted to it.

5 Transformation of HRGs Generating IBHCFLs into HRGs in the Weak Greibach Normal Form

The main theorem we are going to prove is the following:

Theorem 2. *Each ibHCFL L can be generated by an HRG in the WGNF.*

In other words, this theorem states that each context-free hypergraph language satisfying the isolated-node boundedness property can be generated by a grammar in the weak Greibach normal form.

Structurally, the proof of this theorem is based on the corresponding one for string context-free grammars. Let us recall the mains steps of the latter in brief (see details in [1]).

The input of the algorithm is a context-free grammar CFG that does not generate the empty word (this property is related to isolated-node boundedness). The desired output is an equivalent grammar in the weak Greibach normal form (i.e. in which each production is of the form $A \rightarrow aA_1 \ldots A_n$ where a is terminal while $A_1, \ldots, A_n$ are nonterminal).

1. Useless rules and symbols, ε-rules (i.e. rules of the form $A \rightarrow \varepsilon$) and chain rules (i.e. rules of the form $A \rightarrow B$ for A, B being nonterminal) are eliminated.

2. It is shown how to eliminate recursive A-productions, i.e. productions of the form $A \rightarrow A\alpha$ for some fixed nonterminal symbol A. The trick is to move A from the left side of $A\alpha$ to the right side. It is done as follows: if $A \rightarrow A\alpha_1|\ldots|A\alpha_m|\beta_1|\ldots|\beta_p$ are all the A-productions (here | means enumeration of productions), then they are replaced by the productions $A \rightarrow \beta_1 A'|\ldots|\beta_p A'|\beta_1|\ldots|\beta_p$ and $A' \rightarrow \alpha_1|\ldots|\alpha_m|\alpha_1 A'|\ldots|\alpha_m A'$.

3. Left recursion is completely eliminated: nonterminals are numbered as $A_1, \ldots, A_n$ and then a procedure involving application of step 2 is done such that its result is a grammar where each production is either of the form $A_i \rightarrow A_j\alpha$ for $i < j$ or of the form $A_i \rightarrow b\alpha$ for b being terminal.

4. By taking compositions of the above productions and adding new nonterminal symbols the grammar is normalized. The resulting grammar includes rules of the form $A \rightarrow b\alpha$ (b is terminal, α is a string of nonterminals) only.

Our goal is to recreate this proof for HRGs. Note that steps 2-4 of the above plan actively exploit string nature of CFGs: transformations of productions are based on movements of symbols from the leftmost position to the rightmost one. Graphs in general do not have leftmost positions unlike strings, so we are going to distinguish arbitrary edges in graphs so they play the role of "the leftmost symbol". It is done by means of δ (see the proof below).

In the proof we use the following

Lemma 1. *If $\pi = A \rightarrow G(e_0 : B)$ is a production of a grammar $HGr = \langle N, \Sigma, P, S \rangle$ for A, B being non-terminal, and $B \rightarrow H_1, \ldots, B \rightarrow H_k$ are all the productions in HGr with B in the left-hand side, then replacing π by productions $A \rightarrow G[H_1/e_0], \ldots, A \rightarrow G[H_k/e_0]$ does not change the language generated.*

It directly follows from the well-known context-freeness lemma.

5.1 Eliminating Useless, Edgeless and Chain Productions

Our first objective is to eliminate useless productions, productions with no edges in the right-hand side and productions with one nonterminal edge only in the right-hand side. It appears that only this step requires isolated-node boundedness. In the below transformations we provide theoretical reasonings that they can be done irregarding their algorithmic realization; however, the latter can be done similarly to the string case (see [1]).

Definition 10. A nonterminal symbol A in a grammar *HGr* is *useless* if there is no derivation of the form $S \stackrel{*}{\Rightarrow} H(e : A) \stackrel{*}{\Rightarrow} G$ in this grammar where G is terminal.

Transformation 1 (eliminating useless symbols)**.**
Input: an HRG *HGr*.
Output: an equivalent HRG *HGr'* without useless nonterminal symbols.

It suffices to remove all the useless symbols and all the productions containing them. This does not affect the language generated.

Definition 11. A graph is called *edgeless* if it does not contain edges. A production $A \to G$ is called *edgeless* if G is edgeless.

Transformation 2 (eliminating edgeless productions)**.**
Input: an HRG *HGr* that generates an ibHCFL without useless symbols.
Output: an HRG *HGr'* without edgeless productions such that $L(HGr') = L(HGr)$.

Method.
Let $HGr = \langle N, \Sigma, P, S \rangle$. Let $Null = \{(A; H) \mid E_H = \emptyset, A \in N, A \overset{*}{\Rightarrow} H\}$. This set is finite, because otherwise there is a symbol A_0 for which arbitrarily large edgeless graphs H exist such that $(A_0; H) \in Null$; this contradicts the fact that $L(HGr) \in$ ibHCFL (it is important that A_0 is not useless).

   Let $B \to G \in P$ and $E_G = \{e_1, \ldots, e_n\}$. Let P_1 contain the rules of the form $B \to G[H_1/e_1] \ldots [H_n/e_n]$ where $H_i = \odot(lab(e_i))$ for at least one i (then replacement of e_i by H_i changes nothing) and $(lab(e_i); H_i)$ belongs to *Null* otherwise. It is argued that $HGr' = \langle N, \Sigma, P_1, S \rangle$ does not have productions with edgeless right-hand sides (this follows from the construction) and that $L(HGr') = L(HGr)$. $\square$

Definition 12. A production $A \to G$ is called a chain production if $E_G = \{e\}$ and $lab_G(e)$ is nonterminal.

Example 4. The first production in the grammar HGr_2 is chain.

Transformation 3 (eliminating chain productions)**.**
Input: an HRG *HGr* that generates an ibHCFL without useless symbols.
Output: an HRG *HGr'* without chain productions such that $L(HGr') = L(HGr)$.

Method.
Let $HGr = \langle N, \Sigma, P, S \rangle$. Consider the set $Chain = \{(A; H) \mid E_H = \{e_0\}, lab_H(e_0) = B, A, B \in N, A \overset{*}{\Rightarrow} H\}$. Note that *Chain* is finite (again, otherwise one can derive a graph with arbitrarily many isolated nodes having a fixed number of edges). Let $HGr' = \langle N, \Sigma, P'', S \rangle$ where $P' = P \setminus \{A \to H \mid (A; H) \in Chain\}$ and $P'' = P' \cup \{A \to G \mid \exists H : (A; H) \in Chain, H \to G \in P'\}$. Thus we removed all the productions having a graph with one edge in the right-hand side. It can be easily shown that $L(HGr') = L(HGr)$. $\square$

Remark 2. If *HGr* in Transformation 3 does not have edgeless productions, then *HGr'* does not have them either. Transformations 2 and 3 applied to a grammar without useless symbols transform them into a grammar without useless symbols too.

Example 5. The grammar HGr_2 from Example 3 cannot be turned into an equivalent one without edgeless and chain productions.

   The above procedures complete the first step of normalization.

5.2   Defining and Eliminating Recursive Productions

Now we start proving Theorem 2. Let $HGr = \langle N, \Sigma, P, S \rangle$ be a grammar that generates L. Applying Transformations 1, 2, 3, we can assume that *HGr* does not have useless symbols, edgeless productions and chain productions. Thus, the right-hand side of each production has at least two edges or one terminal edge.

   Let us carefully examine what happens in the string case at Step 2. Consider the following

Example 6. Let $A \to Ac|Ad|Be$ be all the A-productions in a string context-free grammar. After Step 2 there are the following ones: $A \to BeA'|Be$, $A' \to c|d|cA'|dA'$. Below we show how a derivation in the old grammar is remodeled in the new one:

Old: $A \Rightarrow Ac \Rightarrow Adc \Rightarrow Addc \Rightarrow Beddc.$

New: $A \Rightarrow BeA' \Rightarrow BedA' \Rightarrow BeddA' \Rightarrow Beddc.$

The underlying idea is to invert the derivation: in the new derivation one starts with Be and applies new productions in reverse order w.r.t. to the old derivation.

The same idea is used in the graph case. The difficulty is that we do not have the leftmost symbol in productions. However, it suffices to distinguish an arbitrary edge in the right-hand side of each production to play the role of the leftmost symbol. We do this by means of the function δ.

Definition 13. Let us fix an arbitrary function δ which acts on P in such a way that $\delta(A \to H)$ belongs to E_H. We denote $\mu(\pi) = lab(\delta(\pi))$.

Now $\mu(\pi)$ plays the role of the "leftmost" symbol of a production π. Then we define recursive productions as expected.

Definition 14. A production π is recursive if $lhs(\pi) = \mu(\pi)$. A production π is an A-production if $lhs(\pi) = A$.

Definition 15. A derivation $A \Rightarrow H_1 \Rightarrow \cdots \Rightarrow H_k$ is called a δ-derivation if each of its productions is applied to the edge that is δ of the previous production. Formally, if $A \underset{\pi_1}{\Rightarrow} H_1 \underset{\pi_2}{\Rightarrow} \ldots \underset{\pi_k}{\Rightarrow} H_k$, then π_i has to be appied to $\delta(\pi_{i-1})$ (in the subgraph that appears after the $(i-1)$-th step). We say that the final label of such a derivation is $\mu(\pi_k)$. Note that $\mu(\pi_i)$ has to be nonterminal for $i < k$ (but $\mu(\pi_k)$ can be terminal).

Below we study how to eliminate all the recursive A-productions for some fixed $A \in N$. In the string case the idea is quite simple: one moves A from the beginning of the string to its end in order to invert a derivation process. Here we also exploit the idea of inverting the derivation but, of course, we have to take into account more complex and general graph structures.

Let $A \underset{\pi_1}{\Rightarrow} H_1 \underset{\pi_2}{\Rightarrow} \ldots \underset{\pi_k}{\Rightarrow} H_k$ be a δ-derivation such that $lhs(\pi_i) = A$ for $i = 1, \ldots, k$ and $\mu(\pi_k) \neq A$ (compare this with the old derivation in Example 6). We provide the following intuition behind the below construction. Imagine that one has a hole in his sweater (this metaphor is related to that in [4]), and he sews it up starting from edges (i.e. he applies $\pi_1, \ldots, \pi_{k-1}$) and finishing by sewing on a patch in the center of the hole (i.e. he applies π_k). In the new derivation one starts with the patch (that is, the right-hand side of π_k), then he sews up right-hand sides of $\pi_{k-1}, \ldots, \pi_2$ in this order (thus the patch grows and becomes bigger) and finishing by connecting it with edges of the hole w.r.t. the production π_1 (see also Fig. 5 which provides a sketch of this process).

The major problem of this idea is the following: if we want to invert a derivation such as on Figure 5a and to obtain a new one, such as one drawn on Figure 5b, we have to carefully control external nodes. Namely, in the old derivation external nodes of H are predefined by R_1 while in the new derivation R_1 appears at the very last step. In order to place external nodes correctly, we introduce complex nonterminal symbols that describe how external nodes of a graph on the current derivation step correspond to external nodes of a graph on the next and on the last step. This description is done by means of partial functions.

Now we proceed with formal realization of this idea. Let $t = type(A)$. Let $\rho_1 = A \to R_1, \ldots, \rho_K = A \to R_K$ be all the recursive A-productions in P and let $\gamma_1 = A \to G_1, \ldots, \gamma_L = A \to G_L$ be the remaining A-productions in P. Our goal is to construct a grammar equivalent to HGr without recursive A-productions. Firstly, we simply remove them: $P_1 := P \setminus \{\rho_1, \ldots, \rho_K\}$. In order to compensate for the lack of these rules we add new ones accordingly to the procedure below.

(a) The production γ.   (b) The production ρ.
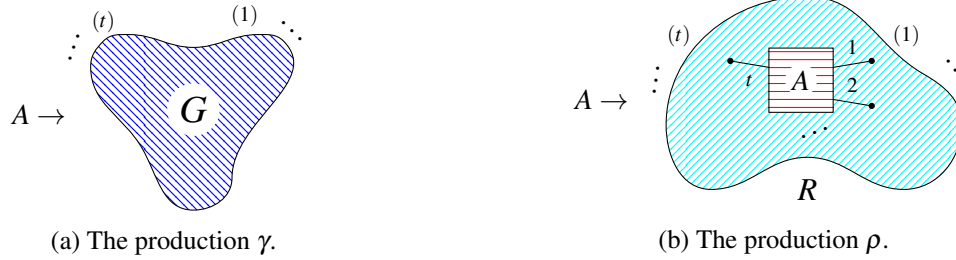
Figure 1: The productions γ and ρ.

Definition 16. $f : X \to Y$ is a partial function if f is a function on some subset X' of X. We denote the domain X' of f by $Dom(f)$, and the range of f by $Ran(f)$.

Definition 17. $f : X \to Y$ is a partial bijection if f is a partial function such that $f|_{Dom(f)}$ is a bijection.

Definition 18. If $f : X \to Y$, $g : Y \to Z$ are partial functions, then $g \circ f : X \to Z$ is a partial function defined on $Dom(f) \cap f^{-1}(Dom(g))$ that acts on this set as a usual composition.

Let $\gamma = \gamma_i$ for some i and let $\rho = \rho_j$ for some j. We define $e_0 := \delta(\rho)$, $\widetilde{e} := \delta(\gamma)$, $G := rhs(\gamma)$ and $R := rhs(\rho)$. Illustrations of these productions are presented in Fig. 1. Firstly, we extend N by *new* nonterminal symbols of the form (A, f, g) where $f, g : [1, t] \to [1, t]$ are two partial bijections; the resulting set is denoted by N'. Then we add to P_1 rules of the following three forms:

Type I. Recall the metaphor about a sweater and a patch. If one imagines an A-labeled edge at the beginning of a derivation as a hole in a sweater, then productions of type I are designed to add a patch G at the very beginning of sewing up (= of a derivation). Partial functions are used to describe what external nodes of G (edges of a patch) are equal to attachment nodes of the A-labelled edge (edges of a hole).

Let $f, g : [1, t] \to [1, t]$ be two partial bijections. Informally, f specifies what nodes of ext_G are external on the second step of the inverted derivation (see again Fig. 5b); g specifies what external nodes of the second step graph are external at the last step. Let $Dom(g \circ f) = \{p_1, \ldots, p_k\}_{IO}$. We set

- $V' := V_G \cup \{u_1, \ldots, u_{t-k}\}$ for $u_1, \ldots, u_{t-k}$ being new nodes;

- $E' := E_G \cup \{e'\}$ where e' is new;

- For $e \in E_G$ we set $att'(e) := att_G(e)$ and $lab'(e) = lab_G(e)$;

- For e' we set $att'(e') := ext_G u_1 \ldots u_{t-k}$;

- $lab'(e') := (A, f, g)$;

- Let $[1, t] \setminus Ran(g \circ f) = \{j_1, \ldots, j_{t-k}\}_{IO}$ (note that $g \circ f$ is bijective so $Dom(g \circ f)$ and $Ran(g \circ f)$ are of the same size). Then we set $ext'(i) := u_r$, if $i = j_r$, $r \in \{1, \ldots, t - k\}$ or $ext'(i) := ext_G(p_r)$ for $i = i_r = g(f(p_r))$, $r \in \{1, \ldots, k\}$.

We are ready to announce that $G' = \langle V', E', att', lab', ext' \rangle$. Finally, we introduce the following rule:

$$\nu_{1, \gamma, f, g} := A \to G'.$$

We define $\delta(\nu_{1, \gamma, f, g}) = \delta(\gamma)$ (note that γ is nonrecursive so is $\nu_{1, \gamma, f, g}$).

Remark 3. Type of (A, f, g) equals $type((A, f, g)) = t + t - k = 2 \cdot type(A) - |Dom(g \circ f)|$.
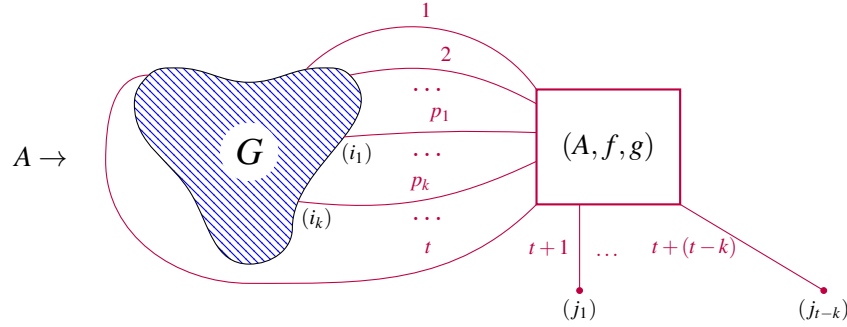
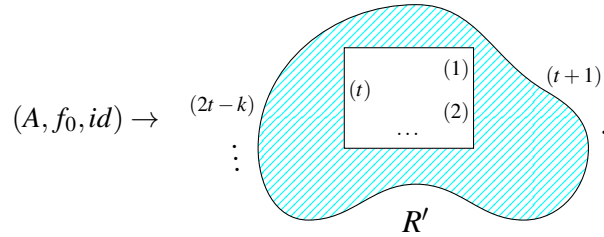Figure 2: The production $\nu_{1,\gamma,f,g}$.

Figure 3: The production $\nu_{2,\rho}$.

Figure 2 illustrates this type of productions.

Type II. Production of type II are used at the last step of a derivation such a on Fig. 5b; returning to the metaphor they allow one to finish sewing up and to finally connect a patch (that has grown by applying productions of types I and III) with the edges of a hole.

We recall that $\rho = A \to R$ such that $\delta(\rho) = e_0 \in E_R$ and $lab(e_0) = A$. Let $f_0 : [1,t] \to [1,t]$ be a function which is defined by the following relation: $f_0(i) = j \Leftrightarrow att_R(e_0)(i) = ext_R(j)$. Obviously, f_0 is a partial bijection. We set

- $V' := V_R$;

- $E' := E_G \setminus \{e_0\}$;

- $att' = att_R|_{E'}$, $lab' = lab_R|_{E'}$

- Let k be equal to $|Dom(f_0)|$ and let $[1,t] \setminus Ran(f_0) = \{j_1, \dots, j_{t-k}\}_{IO}$. Then $ext' = att_R(e_0)ext_R(j_1)$ $\dots ext_R(j_{t-k})$.

We set $R' = \langle V', E', att', lab', ext' \rangle$. Finally, we introduce $\nu_{2,\rho} := (A, f_0, id) \to R'$ and set $\delta(\nu_{2,\rho}) := e$ for some chosen $e \in E_{R'}$ (note that there is one since $|E_G| \geq 2$). Here $id : [1,t] \to [1,t]$ is the identity function. See Figure 3.

Remark 4. $type(R') = 2t - k = 2 \cdot type(A) - |Dom(f_0)| = 2 \cdot type(A) - |Dom(id \circ f_0)| = type((A, f_0, id))$. Thus the production is defined correctly.

Type III. Productions of type III serve to make a "sewing up step" from inside to outside. f,g that specify relations between external nodes on the current step, on the next step and on the final step are changed by related functions f', g'.

Let $f, g, f', g' : [1,t] \to [1,t]$ be partial bijections. The whole procedure described below can be done if $g' \circ f' = g$.

We set

- $V' := V_R \cup \{u_1, \ldots, u_{t-k'}\}$ such that $u_1, \ldots, u_{t-k'}$ are new nodes (k' is defined below);

- $E' := E_R \setminus \{e_0\} \cup \{e'\}$ where $e_0 = \delta(\rho)$ and e' is new;

- For $e \in E_R \setminus \{e_0\}$ we set $att'(e) := att_R(e)$ and $lab'(e) = lab_R(e)$;

- For e' we set $att'(e') := ext_R u_1 \ldots u_{t-k'}$;

- $lab'(e') := (A, f', g')$;

- Let

  - $M_1 := Dom(g) \setminus Ran(f) = \{s_1, \ldots, s_p\}_{IO}$ (nodes of the current step that will be external at the last step but that were not taken into account at the previous step);
  - $M_2 := [1, t] \setminus Ran(g \circ f) = \{j_1, \ldots, j_{t-k}\}_{IO}$;
  - $g(s_i) = j_{l_i}$;
  - Let $M_2 \setminus g(M_1) = \{j_{x_1}, \ldots, j_{x_{t-k-p}}\}$ where $\{x_1, \ldots, x_{t-k-p}\}$ is index-ordered; here we note that $g(M_1) = Ran(g) \setminus Ran(g \circ f)$, thus $g(M_1) \subseteq M_2$.

  Then we set $k' = k + p$ and

  - $ext'(i) := att_R(e_0)(i)$ for $i = 1, \ldots, t$;
  - $ext'(t + l_i) = ext_R(s_i)$ for $i = 1, \ldots, p$;
  - $ext'(t + x_i) := u_i$ for $i = 1, \ldots, t - k'$.

We set $R' = \langle V', E', att', lab', ext' \rangle$. Then $v_{3,\rho,f,g,f',g'} := (A, f, g) \to R'$. We put $\delta(v_{3,\rho,f,g,f',g'}) = e$ for some $e \in E_R \setminus \{e_0\}$. The production is illustrated on Figure 4.

Remark 5. Again, we prove well-definedness of this production:

- Firstly, we check that $type(R') = type((A, f, g))$: $|ext'| = t + p + t - k' = 2t + p - k - p = 2 \cdot type(A) - |Ran(g \circ f)| = type((A, f, g))$;

- Secondly, we have to check that $|att(e')| = type((a, f', g'))$: $|att(e')| = t + t - k' = t + |M_2 \setminus g(M_1)|$. Note that $M_2 \setminus g(M_1) = ([1, t] \setminus Ran(g \circ f)) \setminus (Ran(g) \setminus Ran(g \circ f)) = [1, t] \setminus (Ran(g) \cup Ran(g \circ f)) = [1, t] \setminus Ran(g)$. This yields $|att(e')| = 2t - |Ran(g)| = 2t - |Ran(g' \circ f')| = type((A, f', g'))$.

We say that P_2 is obtained from P_1 by adding all possible productions of all the above types. Obviously, their number is finite (each production is defined by at most four partial functions from $[1, t]$ to $[1, t]$).

Lemma 2. *The grammar* $HGr_1 = \langle N', \Sigma, P_2, S \rangle$ *is equivalent to* $HGr = \langle N, \Sigma, P, S \rangle$.

Proof. Informally, it suffices to notice that the design of new productions allows one to invert productions in the way which is shown on Figure 5. Example 7 provides an illustration of the formal proof, which is given below.

Firstly, we check that $L(HGr) \subseteq L(HGr_1)$. In order to do that, we show how to model a branch of a derivation that consists of rules of the form ρ_i and finishes with a rule γ_j by means of HGr_1. Let $A \underset{\rho_{i_1}}{\Rightarrow} H_1 \underset{\rho_{i_2}}{\Rightarrow} \ldots \underset{\rho_{i_m}}{\Rightarrow} H_m \underset{\gamma_j}{\Rightarrow} H$ be a δ-derivation in HGr where $H_k = H_{k-1}[R_k / \delta(\rho_{i_{k-1}})], 1 \leq k < m$ and $H = H_m[G / \delta(\rho_{i_m})]$. It is convenient to put $H_0 := \odot(A)$ and say that $\delta(\rho_{i_0}) := e_0$ where $E_{H_0} = \{e_0\}$. For the sake of simplyfing notations we write $\rho_1, \ldots, \rho_m$ instead of $\rho_{i_1}, \ldots, \rho_{i_m}$ and γ instead of γ_j.
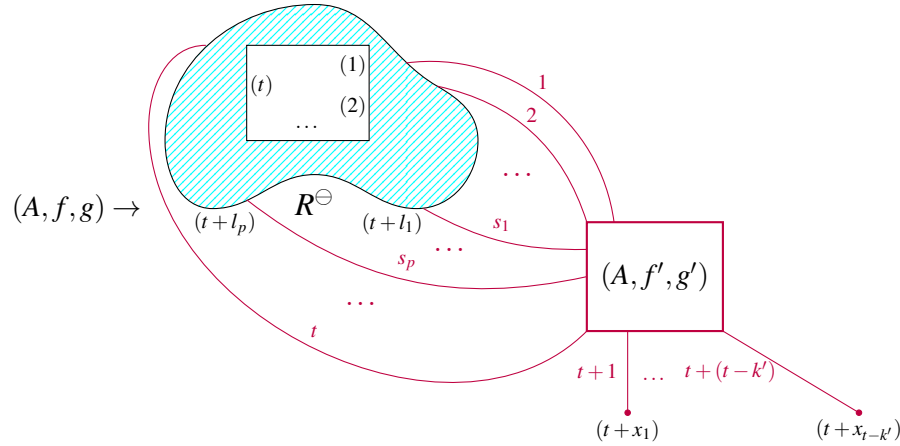
Figure 4: The production $v_{3,\rho,f,g,f',g'}$. Here $R^{\ominus}$ denotes a graph R without $\delta(\rho)$.

Let G_k be defined inductively as follows: $G_{m+1} = G$; $G_k = R_k[G_{k+1}/\delta(\rho_k)]$. Note that

$$
\begin{aligned}
H &= H_m[G/\delta(\rho_m)] = \\
&= H_{m-1}[R_m/\delta(\rho_{m-1})][G/\delta(\rho_m)] = \\
&\quad \dots \\
&= R_1[R_2/\delta(\rho_1)]\dots[R_m/\delta(\rho_{m-1})][G_{m+1}/\delta(\rho_m)] = \\
&= R_1[R_2/\delta(\rho_1)]\dots[G_m/\delta(\rho_{m-1})] = \\
&\quad \dots \\
&= R_1[G_2/\delta(\rho_1)] = \\
&= G_1.
\end{aligned}
$$

To prove the claimed inclusion, it suffices to show then that $A \underset{HGr_1}{\Rightarrow} G_1$.

Observe that $H = H_{i-1}[R_i[G_{i+1}/\delta(\rho_i)]/\delta(\rho_{i-1})]$, $i = 1,\dots,m$. Then G_{i+1} can be considered as a subgraph of $G_i := R_i[G_{i+1}/\delta(\rho_i)]$, which is a subgraph of H. Let f_i be a partial function defined by the following correspondence: $f_i(j) = k \Leftrightarrow ext_{G_{i+1}}(j) = ext_{G_i}(k)$. Similarly we define g_i: $g_i(j) = k \Leftrightarrow ext_{G_i}(j) = ext_H(k)$. It follows from these definitions that $g_i \circ f_i = g_{i-1}$.

Let $\sigma_i = A \to G_{i+1}$ and $G'_i := rhs(v_{1,\sigma_i,f_i,g_i})$. We prove by the reverse induction on $p = m,\dots,1$ that then $A \underset{HGr_1}{\Rightarrow} G'_p$.

Basis. Since $\sigma_m = A \to G = \gamma$ is not a recursive A-production, we added v_{1,σ_m,f_m,g_m} in our grammar. This completes the basis case.

Step. We assume by the induction hypothesis that $A \underset{HGr_1}{\Rightarrow} G'_p$ and our aim is to prove the same for $p - 1$. G'_p contains an edge labeled by (A, f_p, g_p). Then a type III rule is applied to this edge: $v_{3,\rho_p,f_p,g_p,f_{p-1},g_{p-1}}$. The design of such rules allows us to perform exactly the substitution of G_{p+1} into R_p.

Therefore, $A \underset{HGr_1}{\Rightarrow} G'_1$. Note that the right-hand side of this derivation contains an edge labeled by (A, f_1, g_1). Since $G_1 = H$, we see that g_1 is the identity function *id*. Thus we can apply the type II rule of the form v_{2,ρ_1} whose left-hand side equals $(A, f_1, id) = (A, f_1, g_1)$. This rule is also designed in such a way that its application is equivalent to substitution of G_2 into R_1. This means that $A \underset{HGr_1}{\Rightarrow} G'_1 \Rightarrow R_1[G_2/\delta(\rho_1)] = G_1$, as required.

(a) A δ-derivation in *HGr*.

(b) Remodeling the δ-derivation from (5a) by means of HGr_1.
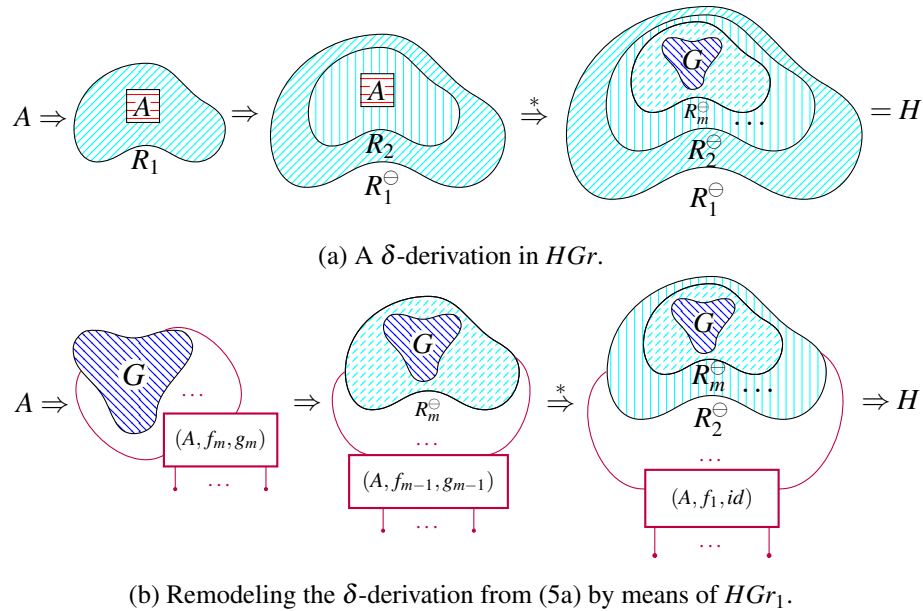
Figure 5: Illustration of the proof of Lemma 2. External nodes are not depicted here.

Now, if there is an arbitrary derivation where the rule ρ_i is applied ($1 \leq i \leq K$), then we find the largest δ-derivation within the derivation that includes this occurence ρ_i and remodel it as described above. Thus, the above reasonings complete the proof for one of the inclusions.

To prove the reverse inclusion, it suffices to show how to transform a branch of a derivation containing nonterminals of the form (A, f, g) into a derivation in *HGr*. Let

$$A \underset{v_1, \gamma, f_m, g_m}{\Rightarrow} G_{m+1} \underset{v_3, \rho_m, f_m, g_m, f_{m-1}, g_{m-1}}{\Rightarrow} G_m \underset{v_3, \rho_{m-1}, f_{m-1}, g_{m-1}, f_{m-2}, g_{m-2}}{\Rightarrow} \ldots \underset{v_3, \rho_2, f_2, g_2, f_1, g_1}{\Rightarrow} G_2 \underset{v_2, \rho_m}{\Rightarrow} G_1$$

be such a branch (it has to be of a similar form). Then we can consider a derivation in *HGr* of the form $A \underset{\rho_1}{\Rightarrow} H_1 \underset{\rho_2}{\Rightarrow} \ldots \underset{\rho_m}{\Rightarrow} H_m \underset{\gamma}{\Rightarrow} H$; the first part of the proof of this theorem allows us to conclude that $H = G_1$. Note that functions f_i and g_i in the first derivation can be arbitrary (they only have to satisfy the condition of type III rules); however, it is not hard to show that they are the same as the functions f_i and g_i that are built on the basis of the second derivation.

This finishes the proof. $\qquad\qquad\square$

There are two important observations regarding the procedure and the lemma above:

1. After such a procedure there are no recursive A-productions;

2. For each nonterminal (A, f, g) there is no rule π such that $\mu(\pi) = (A, f, g)$.

Note that HGr_1 does not preserve all the properties that *HGr* had. Namely, rules of Type II can contain only one edge. However, our further actions do not affect type II and type III rules so this does not matter. Besides, useless symbols can appear, which also does not bother us (we deleted them only to perform Transformations 2 and 3). This finishes the second step.
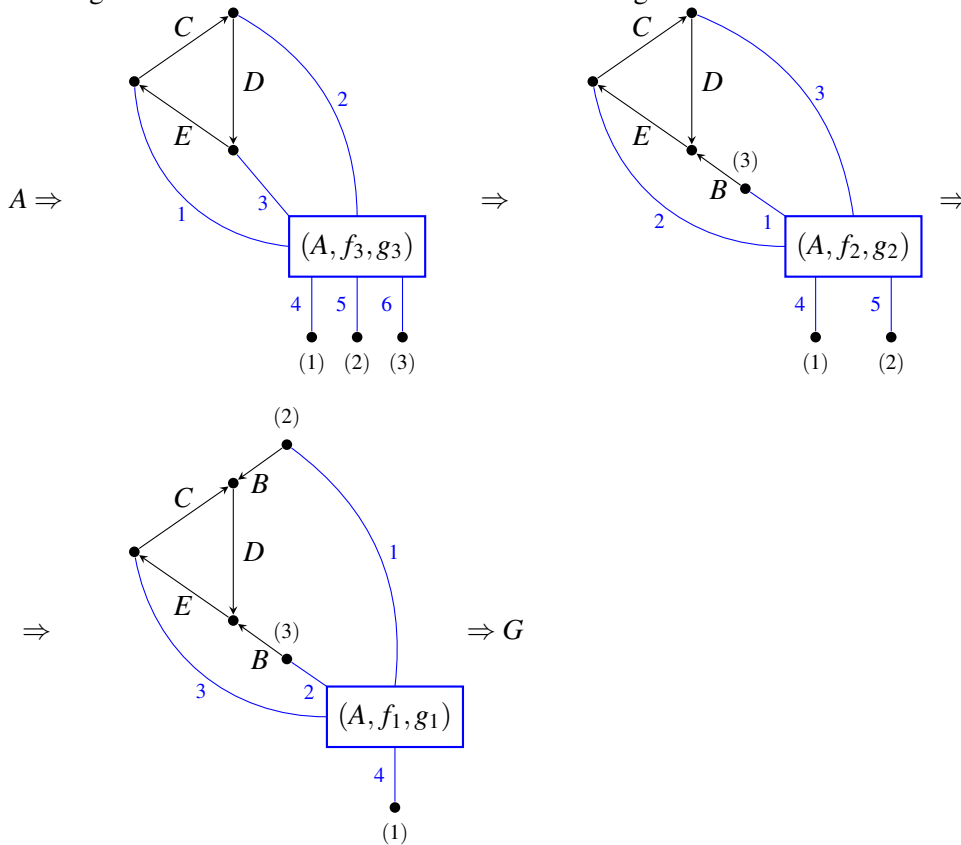
Example 7. Let a grammar contain two A-productions: the first one is recursive and the second one is not recursive. They are depicted below; δ of each production is represented by the red color.
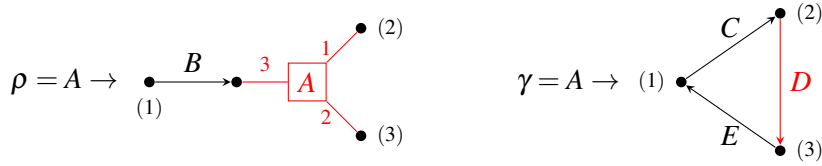
In this grammar the following δ-derivation is possible:
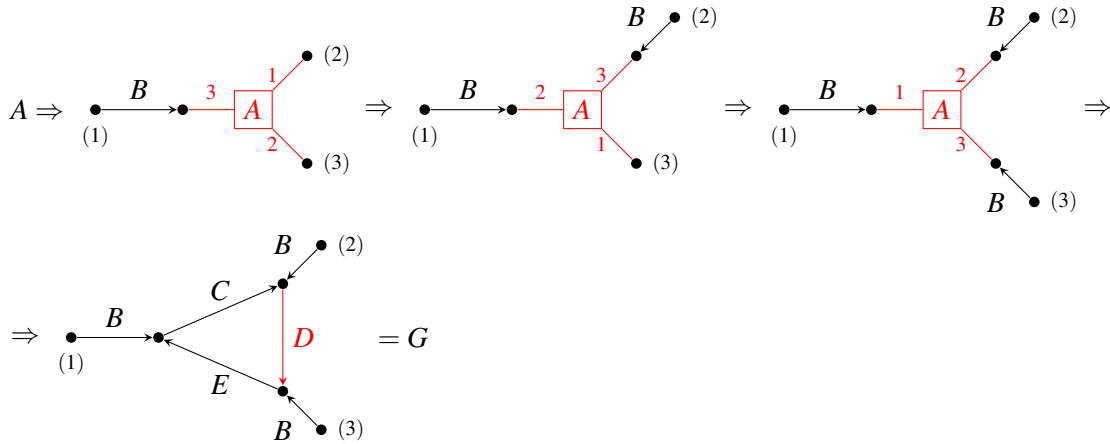
In the new grammar this derivation turns into the following one:

Here $f_3(1) = 2, f_3(2) = 3; f_1 = f_2 = f_3 = g_2; g_3(1) = 3; g_1 = id$ are all the involved partial functions; it is easy to check that they actually satisfy conditions from definitions. Note that if n is the total number of

derivation steps, then f_i specifies a relation between external nodes on the $(n-i)$-th and on the $(n-i+1)$-th steps and it is defined by a corresponding production in the old derivation. In this example, f_i are all defined by the production ρ, thus they are the same. g_i defines a relation between external nodes on the $(n-i+1)$-th step and external nodes at the end of a derivation (on the n-th step). As expected, g_1 is the identity function since $n=4$ and $(n-1+1)=n$, so g_1 specifies a relation between external nodes on the last and on the last step.

5.3 Eliminating Recursion and Convertion into the WGNF

The rest of the steps are similar to those in the string case. Let us start with HGr; let $N = \{A_1, \ldots, A_m\}$. Then the following transformation is done.

Transformation 4 (eliminating recursion).
Input: an HRG HGr as at the beginning of the proof (without useless symbols or edgeless or chain productions).
Output: an equivalent HRG HGr' with the same properties such that for each $\pi \in P$ with $lhs(\pi) = A_i$ either $\mu(\pi) = A_j$ and $i < j$ or $\mu(\pi) \in \Sigma$.

Method.

1. Set $i = 1$.

2. Eliminate all the recursive A_i-productions according to Lemma 2. Now it is argued that if π belongs to the set of productions and $lhs(\pi) = A_i$, then $\mu(\pi) = A_k$ for $k > i$ or $\mu(\pi)$ is terminal.

3. If $i = m$, then a desired grammar is obtained. Otherwise, set $i = i+1$ and $j = 1$.

4. Let $A_j \to H_1, \ldots, A_j \to H_h$ be all the A_j productions. Let $\pi = A_i \to G(e_0 : A_j)$ be a production with $\delta(\pi) = e_0$. Then we can replace π by the rules $A_i \to G[H_1/e_0], \ldots, A_i \to G[H_h/e_0]$ without changing the generated language (see Lemma 1). For these rules we set $\delta(A_i \to G[H_k/e_0]) = \delta(A_j \to H_k)$, $k = 1, \ldots, h$. It will now be the case (due to the previous steps of this transformation) that for each A_j-production π $\mu(\pi)$ equals either a terminal or A_k for $k > j$. Thus all the A_i-productions will have this property after such replacement.

5. If $j = i-1$, go to step 2; otherwise set $j = j+1$ and go to step 4.

$\square$

Let $HGr' = \langle N', \Sigma, P', S \rangle$ be a grammar that is obtained from HGr after Transformation 4. Let $|N'| = M$ (note that M is really large). We already know that HGr' does not have recursions, i.e. there are no δ-derivations $A \underset{\pi_1}{\Rightarrow} H_1 \underset{\pi_2}{\Rightarrow} \ldots \underset{\pi_k}{\Rightarrow} H_k$ such that $\mu(\pi_k) = A$. Indeed, due to the second observation after Lemma 2: nonterminals of the form (B, f, g) cannot participate in recursive δ-derivations since they do not appear as values of μ. For the remaining nonterminals this is a consequence of the result of Transformation 4. Thus we can put a linear order on N' in such a way that for $A \underset{\pi}{\Rightarrow} G(e_0 : B) \in P'$, if $\delta(\pi) = e_0$, then either B is terminal or it is nonterminal and $A < B$.

We define the final set of productions $\overline{P}$ as follows: $A \to G$ belongs to $\overline{P}$ if there is a δ-derivation $A \underset{\pi_1}{\Rightarrow} H_1 \underset{\pi_2}{\Rightarrow} \ldots \underset{\pi_k}{\Rightarrow} H_k = G$ of the length $1 \le k \le M$ such that $\mu(\pi_k)$ is terminal. Obviously, the last requirement yields that G has at least one terminal edge.

Lemma 3. $\overline{HGr} = \langle N', \Sigma, \overline{P}, S \rangle$ *is equivalent to* HGr'.

Proof. Clearly, $L(\overline{HGr}) \subseteq L(HGr')$, because productions in $\overline{HGr}$ are composed of that in HGr'. The reverse inclusion is proved by induction on the length n of derivation in HGr' in more general form: $\langle N', \Sigma, \overline{P}, X \rangle$ is equivalent to $\langle N', \Sigma, P', X \rangle$ for each $X \in N'$.

Basis. If $X \to G \in P'$ then it is also in $\overline{P}$ since G is terminal.

Step. Let $X \to H(e_0 : Y) \overset{n-1}{\Rightarrow} G$ (where $e_0 = \delta(X \to H(e_0 : Y))$) be a derivation of a terminal graph G. This derivation contains a δ-derivation $X \underset{\pi_1}{\Rightarrow} H_1 = H(e_0 : Y) \underset{\pi_2}{\Rightarrow} H_2 \underset{\pi_3}{\Rightarrow} H_3 \ldots \underset{\pi_k}{\Rightarrow} H_k$ and let $\mu(\pi_i) = X_i$ (we also set $X_0 = X$). Then $X_0 < X_1 < \cdots < X_k$, X_k is terminal and all X_i are different; thus $k \leq M$ and $X \to H_k \in \overline{P}$. Then the source derivation is rebuilt as follows: we apply $X \to H_k$ to X and then do the same with nonterminal edges in H_k using the induction hypothesis. This completes the proof. $\qquad\square$

Thus $L(\overline{HGr}) = L$. Now we have to overcome the following problem: the weak Greibach normal form requires right-hand sides of productions to have exactly one terminal edge while we can guarantee for $\overline{HGr}$ that there is at least one terminal edge. This problem is easily fixed: for each $a \in \Sigma$ we introduce a *new* nonterminal symbol T_a and we add rules of the form $T_a \to \odot(a)$. Now, if $A \to G \in \overline{P}$ has more than one terminal symbol in the right-hand side, then we choose one of them and replace the other ones by corresponding nonterminals: $a := T_a$. After this step the modified grammar $\overline{HGr}$ still generates L, and it is in the weak Greibach normal form.

6 Algorithmic Complexity

In the string case polynomial-time algorithms of convertion into the Greibach normal form are known; one is interested whether they exist in the hypergraph case.

It can be seen that the proof we have presented is horrible from the point of view of algorithmic complexity. The main problem is in the second step: the amount of partial bijections from the set $[1,t]$ to itself is equal to $(C_t^0)^2 0! + (C_t^1)^2 1! + (C_t^2)^2 2! + \cdots + (C_t^t)^2 t!$, which grows even faster then any exponential function. It is less obvious but another "heavy" part of the algorithm is the first step — eliminating chain productions. One would hope that presence of these slow parts is the problem of our proof but this is not the case.

Example 8. Consider a grammar $HGr_3 = \langle \{S\}, \{a\}, P_3, S \rangle$ where P has 3 productions ($type(S) = n$, $type(a) = n$):

1. $S \to \langle \{v_1, \ldots, v_n\}, \{e_0\}, att, lab, v_1 v_2 \ldots v_n \rangle$, $lab(e_0) = S$, $att(e_0) = v_2 v_1 v_3 v_4 \ldots v_n$;

2. $S \to \langle \{v_1, \ldots, v_n\}, \{e_0\}, att, lab, v_1 v_2 \ldots v_n \rangle$, $lab(e_0) = S$, $att(e_0) = v_2 v_3 v_4 \ldots v_n v_1$;

3. $S \to \odot(a)$.

Obviously, this grammar has a size $O(n)$. It generates graphs with one a-labeled edge such that attachment nodes of this edge are obtained from external nodes of the graph by a permutation composed of the transposition (12) and of the cycle $(12\ldots n)$. Since these permutations are generators of S_n, the language $L(HGr_3)$ contains $n!$ hypergraphs. All of them have exactly one hyperedge, so for all $H \in L(HGr_3)$ a production $S \to H$ has to belong to a grammar in the WGNF; therefore, even the number of productions grows faster than any exponent apart from the algorithm involved.

This yields the following

Proposition 1. *There is no exponential-time algorithm that takes an HRG generating an ibHCFL and returns an equivalent HRG in the WGNF.*

However, if we consider only HRGs of order r for some fixed r (i.e. in which types of symbols are not greater than r), then the above example ceases to be a problem as well as the first and the second steps in our proof since the number of partial bijections from $[1; r]$ to itself (and, in particular, the size of S_r) is constant when r is fixed.

7 Conclusion

Our desire to introduce the weak Greibach normal form appeared due to another research about hypergraph basic categorial grammars (we presented a paper [10] about them at ICGT-2020); in that work we introduce a new approach to describing graph languages and prove that each grammar in this new formalism can be transformed into an equivalent HRG in the WGNF and vice versa. Theorems 1 and 2 show that ibHCFL is exactly the class of languages generated by grammars in the weak Greibach normal form; thus we also answer a question about the recognizing power of hypergraph basic categorial grammars — they also generate exactly ibHCFL. Besides, we suppose that the issue of the WGNF is a fundamental theoretical question, and we have presented an answer in line with those in [4, 8]. Note that the result gives a natural grammar characterization of the language class ibHCFL.

References

[1] Alfred V. Aho & Jeffrey D. Ullman (1972): *The theory of parsing, translation, and compiling. 1: Parsing*. Prentice-Hall. Available at `https://www.worldcat.org/oclc/310805937`.

[2] Y. Bar-Hillel, C. Caifman & E. Shamir (1960): *On Categorial and Phrase-structure Grammars*. Weizmann Science Press.

[3] Frank Drewes, Hans-Jörg Kreowski & Annegret Habel (1997): *Hyperedge Replacement, Graph Grammars*. In Grzegorz Rozenberg, editor: *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, World Scientific, pp. 95–162, doi:10.1142/9789812384720_0002.

[4] Joost Engelfriet (1992): *A Greibach Normal Form for Context-free Graph Grammars*. In Werner Kuich, editor: *Automata, Languages and Programming, 19th International Colloquium, ICALP92, Vienna, Austria, July 13-17, 1992, Proceedings, Lecture Notes in Computer Science* 623, Springer, pp. 138–149, doi:10.1007/3-540-55719-9_70.

[5] Joost Engelfriet, George Leih & Emo Welzl (1990): *Boundary Graph Grammars with Dynamic Edge Relabeling*. *J. Comput. Syst. Sci.* 40(3), pp. 307–345, doi:10.1016/0022-0000(90)90002-3.

[6] Joost Engelfriet & Grzegorz Rozenberg (1990): *A Comparison of Boundary Graph Grammars and Context-Free Hypergraph Grammars*. *Inf. Comput.* 84(2), pp. 163–206, doi:10.1016/0890-5401(90)90038-J.

[7] Annegret Habel (1992): *Hyperedge Replacement: Grammars and Languages*. *Lecture Notes in Computer Science* 643, Springer, doi:10.1007/BFb0013875.

[8] Christina Jansen, Jonathan Heinen, Joost-Pieter Katoen & Thomas Noll (2011): *A Local Greibach Normal Form for Hyperedge Replacement Grammars*. In Adrian-Horia Dediu, Shunsuke Inenaga & Carlos Martín-Vide, editors: *Language and Automata Theory and Applications - 5th International Conference, LATA 2011, Tarragona, Spain, May 26-31, 2011. Proceedings, Lecture Notes in Computer Science* 6638, Springer, pp. 323–335, doi:10.1007/978-3-642-21254-3_25.

[9] Bevan K. Jones, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann & Kevin Knight (2012): *Semantics-Based Machine Translation with Hyperedge Replacement Grammars*. In Martin Kay & Christian Boitet,

editors: *COLING 2012, 24th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, 8-15 December 2012, Mumbai, India*, Indian Institute of Technology Bombay, pp. 1359–1376. Available at `https://www.aclweb.org/anthology/C12-1083/`.

[10] Tikhon Pshenitsyn (2020): *Hypergraph Basic Categorial Grammars*. In Fabio Gadducci & Timo Kehrer, editors: *Graph Transformation - 13th International Conference, ICGT 2020, Held as Part of STAF 2020, Online, June 25-26, 2020, Proceedings*, Lecture Notes in Computer Science 12150, Springer, pp. 146–162, doi:10.1007/978-3-030-51372-6_9.

[11] Grzegorz Rozenberg, editor (1997): *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, doi:10.1142/3303.