RESTRICTIONS ON NLC GRAPH GRAMMARS*

by

Andrzej Ehrenfeucht[1], Michael G. Main[1],

and Grzegorz Rozenberg[1,2]

CU-CS-258-83 September, 1983

All correspondence to M. Main or G. Rozenberg.

---

* A. Ehrenfeucht and G. Rozenberg were supported in part by National Science Foundation Grant MCS 79-03838. M. Main has been supported in part by a grant from the University of Colorado Council on Research and Creative Work.

(1) Department of Computer Science, Univeristy of Colorado, Boulder CO 80309, USA.
(2) Institute of Applied Mathematics and Computer Science, University of Leiden, The Netherlands.

RESTRICTIONS ON NLC GRAPH GRAMMARS*

Andrzej Ehrenfeucht[1], Michael G. Main[1] and Grzegorz Rozenberg[1,2]

ABSTRACT

Several models of "graph grammars" have been studied with the objective of generating graphs from graphs using a finite set of derivation rules. In this way, possibly infinite sets of graphs (called graph *languages*) can be finitely defined. One aspect that must be addressed in any such model is the "embedding problem" -- that is: When a production is applied, how does the new subgraph get reconnected to the original graph? *Node-label control* (NLC) grammars solve this problem in an elegant way that depends only on the labels of nodes in the new and original graphs. This paper examines certain restrictions on NLC grammars similar to the Chomsky or Greibach normal forms for context-free string grammars. For example, one restriction we consider requires each production to produce a terminal labeled node -- similar to Greibach normal form. We also consider restrictions on the form which the embedding mechanism can take. Our result is that each of the restrictions we examine causes a reduction in generating power for the grammars. Finally, we discuss some directions for future research on NLC grammars.

---

* A. Ehrenfeucht and G. Rozenberg were supported in part by National Science Foundation Grant MCS-79-03838. M. Main has been supported in part by a grant from the University of Colorado Council on Research and Creative Work.

(1) Department of Computer Science, University of Colorado, Boulder, CO 80309, USA.

(2) Institute of Applied Mathematics and Computer Science, University of Leiden, Leiden, The Netherlands.

1. INTRODUCTION

Graph grammars provide a mechanism for generating sets of graphs. A survey of these graph grammars has been given by Nagl, together with a list of numerous applications such as pattern recognition, semantics of programming languages, data flow analysis and code optimization [9]. A procedure for transforming graphs which is common to most "sequential" graph grammars is described as follows [9,10] (by "graph" we refer to an undirected, node-labeled, finite graph).

Assume we have a graph which we want to transform using a production $\alpha \rightarrow \beta$, where α and β are graphs. Then one follows these four steps:

(1) Locate some instance of a particular subgraph α in the graph to be transformed;

(2) Delete this instance of the subgraph α;

(3) Introduce a new subgraph β;

(4) Using some fixed algorithm, embed the new subgraph β in the remaining nodes of the graph.

Thus, there is a major difference between applying a graph production $\alpha \rightarrow \beta$ and applying the analogous string production: for a string production, there is no need to specify how the new substring is to be embedded in the original string. Hence, an important facet of any model of graph grammars is the method for embedding an introduced subgraph into the nodes of the original graph.

A model of graph grammars introduced in [4] provides a simple solution to the embedding problem: the embedding of an introduced subgraph is entirely controlled by labels of node. These grammars are called *node-label controlled* graph grammars (abbreviated NLC grammars), and the sets of graphs which they generate are NLC *languages*. A number of properties of these graph languages have been recently studied [5,6,7,8].

This paper sets several restrictions on NLC grammars and examines the generative power of the resulting subclasses of grammars. The restrictions are of two types:

(1) Restrictions on the sort of productions which may be used.

(2) Restrictions on how an introduced subgraph may be embedded in the original graph.

Restrictions of the first type are familiar in string grammars -- for example, the restriction of context-free grammars to Chomsky normal form [1,3,11] or Greibach normal form [2,3,11].

The first restriction we consider is based on Chomsky normal form. A key property of a Chomsky normal form grammar is that the right side of any production has length two or less. The usefulness of this form is that any context-free language can be generated by such a grammar. The similar restriction that we place on NLC grammars is to limit the right side of any production to a graph with at most k nodes, where $k \geq 2$ is some fixed integer. Such a grammar is called k-ary. We show that this restriction reduces the generating power of NLC grammars. In fact, for any k, there are finite languages which no k-ary grammar generates. Directly from this result we can demonstrate certain simple infinite languages that no NLC grammar generates.

The second production restriction we consider is based on Greibach normal form for context-free string grammsrs. In a Greibach normal form grammar, the right side of each production always consists of a single terminal symbol followed by a string of nonterminals. Once again, every context-free language is generated by some Greibach normal form grammar. Such a grammar is useful in parsing a language or building a pushdown automaton to accept the language. The similar restriction we place on NLC grammars is to require the right side of each production to contain at least one terminal-labeled node. We show that this restriction reduces the generating power of NLC grammars. That is, there are NLC languages which cannot be generated by a grammar with

this restriction.

Finally, we consider three restrictions of the second type -- *i.e.*, restricting the embedding mechanism. Two of these restrictions capture an ingredient of generative determinism which has no direct counterpart in string grammars, but which could be useful in developing parsing algorithms. The third restriction introduces an element of symmetry in the embedding mechanism, which would be useful in extending the NLC model to parallel rewrite systems. These three restrictions which we consider all result in a reduction in the generative power of NLC grammars.

NOTATION: If Σ is a finite alphabet, then G_Σ denotes the set of all (finite, undirected) graphs with node labels from Σ. If α is a graph and v is a node of α with label X, then we call v an X-node. The number of nodes in α is denoted by $|\alpha|$.

2. NLC GRAMMARS

A *(graph) language* is a set of graphs with node labels from some fixed finite set of symbols. Graph languages can be generated using graph grammars, and in particular using NLC grammars. The production rules of an NLC grammar are similar to those of a context-free string grammar; the embedding component is specified by means of a *connection relation.* Formally, an NLC grammar is defined as follows:

Definition: An NLC grammar is a five-tuple (Σ,Δ,P,S,C), where:

Σ is a finite set of *labels.*

Δ is a proper subset of Σ, called *terminal labels.*

P is a finite set of *productions;* each production has the form $X \to \alpha$, where X is a nonterminal label ($X \in \Sigma - \Delta$) and α is a graph from G_Σ.

S is a special nonterminal called the *start symbol.*

C is a binary relation $C \subseteq \Sigma \times \Sigma$, called the *connection relation*. □

This definition of an NLC grammar is more general than the original [4] in that it allows the right side of a production to be the empty graph. However, this is an inessential difference.

Let $G = (\Sigma, \Delta, P, S, C)$ be an NLC grammar. The way that a production $X \to \alpha$ of P is applied to transform a graph is as follows:

(1) Start with a graph μ and a specific occurrence of an X-node in μ. We call this node the *mother* node. The set of nodes which are directly connected to the mother node is called the *neighborhood*.

(2) Delete the mother node from the graph μ, and call the resulting graph μ'.

(3) Add to μ' a copy of the labeled graph α. This new occurrence of α is called the *daughter* graph.

(4) For each pair (Y,Z) in the connection relation, connect *every* Y-node in the daughter graph to *every* Z-node in the neighborhood.

Call the resulting graph η. We write $\mu \underset{G}{\Rightarrow} \eta$ to denote the relation "η is *directly derived* from μ in G".

If there exists a finite sequence of transformations:

$$\mu_0 \underset{G}{\Rightarrow} m_1 \underset{G}{\Rightarrow} \cdots \underset{G}{\Rightarrow} \mu_m$$

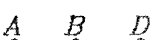
then we write $\mu_0 \overset{*}{\underset{G}{\Rightarrow}} \mu_m$ and say that μ_m is *derived from* μ_0 in G. The finite sequence is called a *derivation of length m*. When G is understood, we will simplify the notation to $\mu \overset{*}{\Rightarrow} \eta$ or $\mu_0 \overset{*}{\Rightarrow} \mu_m$.

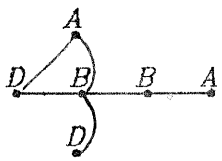
The *language generated by the grammar G*, also called an NLC *language*, is the set of all graphs with terminal labels, which can be derived from the one-node graph with

label S. That is:

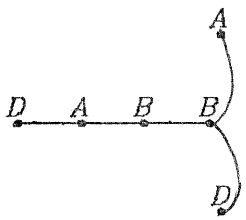
$$L(G) = \{\mu \in G_\Delta \mid S \overset{*}{\underset{G}{\Rightarrow}} \mu\}.$$

(Here we have used S to denote the one-node graph with label S.)

Example: Suppose we have a production $A \to$  in an NLC grammar with connection relation $C = \{(A,D),(B,B),(B,D)\}$. We could apply this production to the graph  in two different ways. One possibility is to remove the leftmost A-node and replace it by the subgraph  . Embedding this subgraph according to the connection relation C gives this result:



Another possibility is to apply the production to the rightmost A-node. This results in the graph:



□

Example: Consider $G = (\Sigma,\Delta,P,S,C)$, where $\Sigma = \{S,a\}$, $\Delta = \{a\}$, $C = \Sigma\times\Sigma$ and P contains these two productions:

$S \to$  $\qquad S \to$ 

Clearly $L(G)$ consists of all nonempty complete graphs with every node labeled a. If we

change the connection relation to contain only the pair (a,a), then the language becomes the set of nonempty chains with every node labeled a. □

We finish this section with some preliminary results on NLC languages.

Definition: For any integer $k \geq 0$, an NLC grammar is called *k-ary* provided that each production $X \to \alpha$ has $|\alpha| \leq k$. An *erasing production* is a production $X \to \lambda$, where λ is the empty graph. □

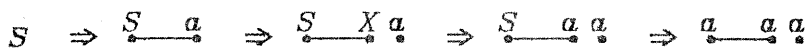
Erasing productions for graph grammars are similar to erasing productions for context-free string grammars [3,11] – and like the string grammars, erasing productions may always be eliminated from an NLC grammar (unless λ itself is being generated). In fact, these productions can be eliminated without increasing the arity of the grammar:

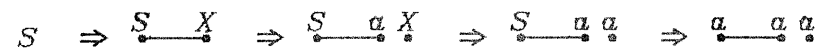
Theorem 2.1. *Let $k \geq 0$ be an integer and $G = (\Sigma,\Delta,P,S,C)$ be a k-ary grammar. Then there exists a k-ary grammar $G' = (\Sigma,\Delta,P',S,C)$ such that P' contains no erasing productions and $L(G') = L(G) - \{\lambda\}$.* □

The proof of this theorem is virtually identical to the corresponding proof for context-free string grammars [11, Theorem 6.2].

In general, there may be several different derivations of a particular graph in a given grammar. For example, consider the grammar with $\Sigma = \{S,X,a\}$, $\Delta = \{a\}$, C containing (a,a) and (a,S), and these four productions:

Here are two derivations of the same three-node graph:





The first of these derivations has the following property: at the last point in the derivation where the graph has fewer than three nodes, there is exactly one nonterminal label. Such situations are useful in analyzing derivations, and this leads us to the following notion:

Let μ be a graph with $|\mu| = n > 1$. A derivation of μ in an NLC grammar is called *constrained* if at the last point in the derivation where the graph has fewer than n nodes, there is exactly one nonterminal. If a graph is derivable in some grammar, there may or may not be a constrained derivation of it. (Compare this to the situation for string grammars.) However, the following theorem does guarantee that some constrained derivations exist.

Theorem 2.2. *Let μ be a graph with $|\mu| > 1$, and let G be an NLC grammar with no erasing productions. If $\mu \in L(G)$, then there exists a graph η such that η has a constrained derivation in G and η differs from μ only in its edges.*

Proof: The proof consists of a modification of a derivation of μ to achieve a constrained derivation. The modification may change the edges in the final graph, but it will not change the number or labels of nodes. Begin with a derivation of μ:

$$S \Rightarrow \mu_1 \Rightarrow \cdots \Rightarrow \mu_k \Rightarrow \mu$$

We focus our attention on the last graph in this sequence with fewer than $|\mu|$ nodes, and call this graph the *critical graph*. At this point, a production, say $X \rightarrow \alpha$, is applied to a particular X-node which we will call e. After this point, the only sort of productions which are applied are relabeling of nodes -- since any other sort of production increases the number of nodes beyond $|\mu|$. So, a new derivation is given in this way:

(1) Up to the critical graph, the modified derivation is identical to that of μ.

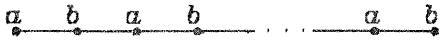
(2) At this point we skip the rewriting of the node e. Instead, we look at each of the nodes other than e and apply productions from the latter part of the μ derivation to bring the labels to terminals.

(3) Finally, to the node e, we apply the production $X \to \alpha$, creating some new nodes. By applying productions from the latter part of the μ derivation we can bring all the labels of α to terminals.

This is obviously a constrained derivation and the resulting graph is identical to μ, except for its edges. □

Corollary 2.3. *Let G be an NLC grammar with no erasing productions and let $n>1$ be an integer. If G generates exactly one graph μ with $|\mu|=n$, then there is a constrained derivation of μ in G.* □

Theorem 2.4. *Let G be an NLC grammar and suppose that for every integer $m \geq 0$, there exists a graph $\mu \in L(G)$ such that μ has the subgraph:*



(2m nodes)

Then either (a,b) or (b,a) is in the connection relation of G.

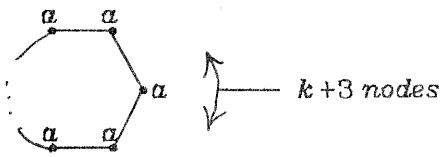
Proof: The proof is by contradiction. Suppose neither (a,b) nor (b,a) is in the connection relation of G. Then the only way a subgraph of the given form can be constructed is if it appears whole in the daughter graph of some production. Since each daughter graph is finite, this implies an infinite number of productions -- but the number of productions is also finite. □

3. RESTRICTIONS ON PRODUCTIONS

We have already introduced one restriction on productions: the k-ary grammars. Recall that an NLC grammar is called k-ary provided that every production $X \to \alpha$ has $|\alpha| \leq k$. The first result of this section shows that for every k, the k-ary grammars are

less powerful than general NLC-grammars:

Theorem 3.1. *Let $k>1$ be an integer and define μ to be the following graph:*

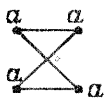


Any k-ary grammar which generates μ must also generate another graph with $k+3$ nodes.

Proof: Assume to the contrary that G is a k-ary NLC grammar and μ is the only $k+3$ node graph G generates. By Theorem 2.1 we may assume that G contains no erasing productions, hence by Corollary 2.3 there must be a constrained derivation of μ. At some point in this constrained derivation there is a step $\beta \Rightarrow \gamma$ with $|\beta| < k+3$ and $|\gamma| = k+3$. Consider the daughter graph α which has been inserted at this step, and its corresponding neighborhood (*i.e.*, the neighborhood of the mother node). Each node in the neighborhood is already labeled a (since the derivation is constrained), and eventually each node in α will be relabeled with a. The key observations are these:

- The neighborhood must contain at least two nodes, for otherwise the final graph cannot be of the required form (*i.e.*, it won't contain a Hamiltonian cycle).
- At least two nodes in α must remain connected to some node in the neighborhood, for otherwise the final graph cannot contain a Hamiltonian cycle.
- Any node in α which remains connected to part of the neighborhood must remain connected to *all* of the neighborhood, because every node in the neighborhood is labeled identically.

From these three observations, it follows that the final graph has a subgraph like this:

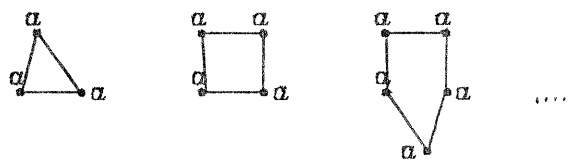


from the neighborhood from α

But no such subgraph occurs in μ. By this contradiction we conclude that every k-ary grammar that generates μ also generates some other $k+3$ node graph. □

Corollary 3.2. *For every $k \geq 0$, there are NLC languages that are not generated by any k-ary NLC grammar.* (In particular, the finite language containing only the graph μ of Theorem 3.1 is NLC but not k-ary.)

Corollary 3.3. *Consider the infinite sequence of graphs:*



No infinite subset of this sequence is an NLC language. □

The second restriction on NLC grammars that we consider is to require each production to produce at least one terminal.

Definition: An NLC grammar is called *positive* if for every production $X \rightarrow \alpha$, the graph α contains at least one terminal label. □

Obviously any NLC language containing the empty graph cannot be generated by a positive grammar. We will show that there are also NLC languages without the empty graph which are not generated by any positive grammar. The method is as follows: Let $k>1$ be an integer. We will define a graph τ_k which is not generated by any k-ary positive grammar. The graph τ_k contains $6(k+1)$ nodes which we will denote by triples $\langle x,y,z \rangle$ with $x \in \{1,2\}$ and $y \in \{1,2,3\}$ and $z \in \{1,2,\ldots,k+1\}$. Each node in τ_k is labeled by 'a', and a

node $<x_1,y_1,z_1>$ is connected to another node $<x_2,y_2,z_2>$ iff $x_1=x_2$ or $y_1=y_2$ or $z_1=z_2$. Note these properties that τ_k has:

Property 1: Among any three nodes in τ_k, at least two are connected (since the x-coordinate has only two possible values).

Property 2: Every node in τ_k is disconnected from exactly $2k$ other nodes.

Property 3: For any two nodes of τ_k, the number of other nodes from which they are *mutually disconnected* is at most $2k-2$.

Theorem 3.4. *Let $k>1$ be an integer. The graph τ_k is not generated by any k-ary positive grammar.*

Proof: Suppose to the contrary that G is a k-ary positive grammar generating τ_k. The connection relation of G must contain (a,a), otherwise G cannot generate any connected graph with more than k nodes. Consider a derivation of τ_k in the grammar G. Somewhere in this derivation, the number of nodes increases from some number less than $6(k+1)$ to exactly $6(k+1)$. We will focus on one of the new terminal nodes created at this step of the derivation. This new terminal node is labeled by a, and subsequent steps in the derivation cannot disconnect it from any other node. (This is because subsequent steps may only relabel an existing node with a, and (a,a) is in the connection relation.) Thus, at the time of its creation, the new terminal node is disconnected from exactly $2k$ other nodes (by property 2). Where could these $2k$ nodes be? At least $k+1$ of them must be outside the new daughter graph, since this daughter graph has at most k nodes. We will call these the "outside disconnected nodes", and consider these two cases for their location:

Case 1: In this case, two or more of the outside disconnected nodes are in the neighborhood of the new daughter graph. Let X and Y be the labels of two such outside disconnected nodes. Neither X nor Y is equal to a, since (a,a) is in the

connection relation. So, eventually, we will have to change both the X and the Y to label a. But this will disconnect these two outside nodes from each other, since neither (a,X) nor (a,Y) is in the connection relation. This creates three mutually disconnected nodes, which contradicts property 1.

Case 2: In this case, one or zero of the outside disconnected nodes are in the neighborhood of the daughter graph. This implies that at least one of the outside disconnected nodes is not in the neighborhood. Notice that such a node must be disconnected from every node in the new daughter graph. This implies that any two nodes in the daughter graph are connected to each other (by property 1). Hence, there must be exactly $2k$ outside disconnected nodes. At least $2k-1$ of these are not in the neighborhood. These $2k-1$ nodes must all be disconnected from the entirity of the new daughter graph. But this contradicts property 3.

By these contradictions, we conolude that G cannot generate τ_k. □

Corollary 3.5. *Let Δ be any nonempty alphabet. The NLC language $G_\Delta - \{\lambda\}$ consisting of all nonempty graphs over Δ is not generated by any positive grammar.* □

4. RESTRICTIONS ON THE CONNECTION RELATION

An NLC grammar $G = (\Sigma,\Delta,P,S,C)$ is called *functional* if its connection relation $C \subseteq \Sigma\times\Sigma$ is a partial function -- *i.e.*, for every $X \in \Sigma$ there is at most one $Y \in \Sigma$ with $(X,Y) \in C$. The grammar is called *inverse functional* if the inverse of the connection relation is a partial function -- *i.e.*, for every $Y \in \Sigma$ there is at most one $X \in \Sigma$ with $(X,Y) \in C$. The grammar is called *symmetric* if the connection relation is symmetric -- *i.e.*, whenever (X,Y) is in C, then (Y,X) is also in C. In this section we show that all three of these restrictions reduce the generating power of NLC grammars.

The case of functional and inverse functional grammars are treated simultaneously using the language $G_{\{a,b\}}$ of all graphs with terminal labels $\{a,b\}$. It is easy to give an NLC grammar for this language. Thus, the following theorem shows that the functional and inverse functional restrictions reduce the generating power of NLC grammars:

Theorem 4.1. *The NLC language $G_{\{a,b\}}$ is not generated by any functional or inverse functional NLC grammar.*

Proof: The language contains arbitrarily long chains of the form $a\ a\ \cdots\ a$ and $b\ b\ \cdots\ b$. By Theorem 2.4, this implies that both (a,a) and (b,b) are in the connection relation of any grammar for $G_{\{a,b\}}$. The language also contains arbitrarily long chains of the form

$$a\ b\ a\ b\ \cdots\ a\ b$$

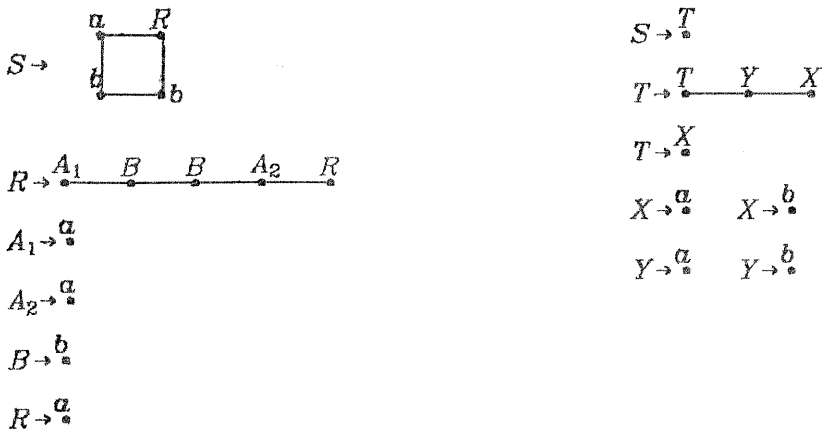
Again, by Theorem 2.4, this implies that either (a,b) or (b,a) is in the connection relation of any grammar for $G_{\{a,b\}}$. Either of these choices (that is (a,b) or (b,a)) makes the connection relation neither functional nor inverse functional. □

The case of symmetric grammars needs a more complex language. We define a language L with terminal alphabet $\{a,b\}$. The language contains these two sorts of graphs:

(1) Any odd length chain, for example $a\ b\ b$ or $a\ b\ a\ a\ b$.

(2) Any graph with $4m$ nodes ($m \geq 1$) in a circle. If read with the proper orientation the labels of these nodes must be $aabbaabb\ldots$, like this:

It is not difficult to construct a non-symmetric grammar for L. The nonterminal label set is $\Sigma = \{S,T,X,Y,R,A_1,A_2,B\}$ with S as the start symbol. The productions are these:



The connection relation contains any pair where the first component is a,b or X, plus the pairs (A_1,a), (A_1,A_2) and (R,b). Any derivation that starts with the production $S \to \overset{T}{\bullet}$ will derive an odd length chain. The other derivations generate the cyclic graphs of L. Thus, L is an NLC language; but the following theorem shows it is not generated by any symmetric grammar.

Theorem 4.2. *No symmetric NLC grammar generates the NLC language L.*

Proof: Suppose to the contrary that G is a k-ary symmetric grammar generating L ($k>0$). We will need some properties concerning G. First, by Theorem 2.1 we may assume that G has no erasing productions. Second, by Theorem 2.4 we know that (a,a), (b,b) and at least one of (a,b) or (b,a) are in the connection relation of G. But, since G is symmetric, this means that all of $\{(a,a),(a,b),(b,a),(b,b)\}$ is in the connection relation, that is:

Terminal Property: Any two terminals are related by the connection relation.

The rest of the proof deals with the unique graph in L with $4k$ nodes. We call this graph σ and note that it has the following property:

Disconnection Property: Let μ be any subgraph of σ with $2 \leq |\mu| \leq k$. Then every node in μ is *disconnected* from at least one neighbor of μ. (The neighbors of μ are the nodes which are not in μ, but which are directly connected to some node of μ.)

Now, consider a derivation of σ. By Theorem 2.3 we may assume this is a constrained derivation. At some point in the derivation the number of nodes increases from some number less than $4k$ to exactly $4k$. We will focus on the new daughter graph created at this point. Since the derivation is constrained, the neighborhood of this daughter graph contains only terminal nodes. But, by the disconnection property, every node in the daughter graph must disconnect from at least one node in the neighborhood. This implies that every node in the daughter graph is initially labeled with a nonterminal (by the terminal property).

Consider some particular node in the daughter graph. Eventually this must be labeled with a nonterminal X which causes the node to disconnect from a neighbor node. Without loss of generality, we may assume this neighbor node is labeled b -- *i.e.*, neither (X,b) nor (b,X) is in the connection relation. But, suppose after the daughter graph is created we apply these productions:

(1) Change the label of the particular daughter node to X (or leave it alone if it is already X).

(2) Change every other label to a terminal.

(3) Change the X-node to a terminal label.

This results in a $4k$ node graph that contains a node which is not connected to any b-node. Such a graph doesn't occur in L, and by this contradiction we conclude that no symmetric grammar generates L. □

5. DISCUSSION

We have shown that a number of restrictions on NLC grammars reduce the generative power. In particular, restrictions which correspond to "Chomsky normal form" and "Greibach normal form" are not normal forms for the NLC grammars. The question remains as to what "useful" restrictions do not reduce the generating power -- that is, what normal forms exist for NLC grammars? Particularly desirable are normal forms that would aid in developing parsing techniques.

A question that remains unanswered is the power of functional, inverse functional and symmetric grammars for a one-letter terminal alphabet. (Our results apply only to alphabets of size two or more). A second question to be investigated is exactly how the classes of languages generated by the restricted grammars differs from arbitrary NLC languages. We are particularly interested in the class of languages generated by symmetric grammars because such a restriction is a natural choice when extending NLC grammars to parallel rewrite systems.

References

(1) N. Chomsky. On certain formal properties of grammars, *Information and Control* 2 (1959), 137-167.

(2) S.A. Greibach. A new normal form theorem for context-free phrase structure grammars, *JACM* 12 (1965), 42-52.

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

(4) D. Janssens and G. Rozenberg. On the structure of node-label controlled graph languages, *Information Sciences* 20 (1980), 191-216.

(5) D. Janssens and G. Rozenberg. Restrictions, extensions and variations of NLC grammars, *Information Sciences* 20 (1980), 217-244.

(6) D. Janssens and G. Rozenberg. Decision problems for node-label controlled graph grammars, *JCSS* 22 (1981), 144-177.

(7) D. Janssens and G. Rozenberg. A characterization of context-free string languages by directed node-label controlled graph grammars, *Acta Informatica* 16 (1981), 63-85.

(8) D. Janssens and G. Rozenberg. Graph grammars with neighbourhood controlled embedding, *TCS* 21 (1982), 55-74.

(9) M. Nagl. A tutorial and bibliographical survey on graph grammars, in: *Graph-Grammars and Their Application to Computer Science and Biology* (V. Claus, H. Ehrig and G. Rozenberg, eds), LNCS 73, Springer-Verlag, Berlin (1978), 70-126.

(10) B.K. Rosen. Deriving graphs from graphs by applying a production, *Acta Informatica* 4 (1975), 337-357.

(11) A. Salomaa, *Formal Languages*, Academic Press, New York, 1973.