ELSEVIER
Theoretical Computer Science 207 (1998) 25–41
Theoretical Computer Science

The ancestor width of grammars and languages

Franz J. Brandenburg *
Lehrstuhl für Informatik, Universität Passau, 94030 Passau, Germany

Abstract

The ancestor width is a new measure for the structure of derivations of arbitrary grammars. For every production used in a derivation or equivalently for every leaf we consider the strings of ancestors. The ancestors define a complexity measure with a local flavour. Obviously, context-free grammars have ancestor width one. We show that languages with ancestor width two are context-free. However, every recursively enumerable set can be generated by a grammar with ancestor width three. For λ-free grammars the ancestor width is closely related to nondeterministic space complexity. Then languages such as $\{wcwc|w \in \{a,b\}^*\}$, $\{a^n b^n c^n | n \geq 1\}$ or $\{(a^n c)^n | n \geq 1\}$ can be generated with ancestor width four. Moreover, any language can be generated with ancestor width three, if padding is used and the language is represented with tails.

Keywords: Grammars and complexity; Non-context-free languages; Structure of derivations

1. Introduction

The theory of formal languages is one of the fundamental areas of Theoretical Computer Science. It has been investigated since the 1960s. However, most of the research effort has been directed towards the theory of context-free grammars and languages. The theory of context-sensitive and phrase structure grammars is not well developed. These grammars generate highly complex languages, which are more commonly described by machines and studied in terms of their computational complexity, such as time and space.

In his early research work, Professor Ron Book investigated context-sensitive grammars. He transferred the notion of time complexity to grammars [2, 3, 7] and made attempts to understand the use of context in derivations of context-sensitive grammars. His "connectivity lemma" [2, 3] sheds some light on the structure of context-sensitive derivations. Barriers of terminal context impose structural properties of a different type [4]. Then context-sensitive grammars generate only context-free languages [1]. In [5] Book writes "*The way in which rewriting rules containing context interact to generate*

* E-mail: brandenb@informatik.uni-passau.de.

0304-3975/98/$19.00
PII S0304-3975(98)00054-1

non-context-free languages has not been explained. While it is undecidable whether a context-sensitive (or arbitrary) grammar generates a context-free language, it is not unreasonable to ask for a formal description of the mechanism used in derivations of such grammars in order that non-context-free languages are generated. In many examples it appears that a capacity for "storing and transmitting information" is coded into the rewriting rules by choice of context. Several results have used the creation of "barriers" to "message-sending" in order to show that under certain conditions only context-free languages are generated. The description of the types of rules used in showing that a grammar can imitate the actions of a Turing machine does little to explain the mechanism (or power) of context."

In this paper we define a new measure on the derivations of grammars. For every left-hand side of a production or equivalently for every symbol of the generated string and every derived occurrence of the empty string we consider the strings of ancestors, which are the predecessors in the derivation. This relation is based on the fact, that the full left-hand side α of a production $\alpha \to \beta$ is the string of ancestors of every symbol of β. Obviously, a production is context-free if and only if it has a string of ancestors of length one. Hence, context-free grammars and languages have ancestor width one. This can be interpreted by saying that "*context-freeness is for free*".

In a derivation of an arbitrary grammar, every substring of a derived string depends only on its strings of ancestors. The strings of ancestors completely describe the history in the derivation. Derivations can be rearranged into an initial part on the strings of ancestors towards the distinguished substring and two independent left and right subderivations. By recursion, every derivation of an arbitrary grammar can be transformed into a syntactically equivalent derivation with a global tree structure, where the vertices consist of local subderivations on strings of ancestors, see [9–12].

The ancestor width measures the size of the strings of ancestors of the left-hand sides of the productions, or equivalently of the leaves of the derivation. It does not suffice to consider only the symbols of the generated string. This notion has been introduced in [10] and has been investigated for context-sensitive grammars and languages in [9]. Here, we consider arbitrary grammars. Now, λ-productions generating the empty string play a crucial role. We show that every recursively enumerable set can be generated by a grammar with ancestor width three. Thus arbitrary languages can be generated by spreading out information over strings of length three. However, grammars with ancestor width two generate only context-free languages. For λ-free grammars, the ancestor width is closely related to the workspace of grammars, which coincides with the space complexity of nondeterministic Turing machines. Such grammars can generate e.g. the languages $\{a^n b^n c^n | n \geq 1\}$ and $\{wcwc | w \in \{a, b, \}^*\}$ with ancestor width four. Moreover, any language L can be generated by a λ-free (resp. context-sensitive) grammar with ancestor width four (resp. three), if padding is used and the language is represented with tails. The length of the tails is related to the time and space complexity of L.

This research was motivated by Professor Book's research on grammars [2–5, 7] and was stimulated by his encouragements and useful hints while I was writing my dissertation [10].

2. Preliminaries

We review some basic definitions on strings and grammars. A *string* $w = a_1..a_n$ is a finite sequence of symbols from an alphabet Σ. Its length is denoted by $|w|$. The *substring* $a_i..a_j$ of w from the i-th to the j-th symbol is denoted by $w(i,j)$, where $w(i,j) = \lambda$, if $j < i$. Here λ denotes the empty string.

Definition 1. A (phrase-structure or Type 0) grammar is a quadruple $G = (N,T,P,S)$, where N and T are the alphabets of nonterminal and terminal symbols with $N \cap T = \emptyset, S \in N$ is the axiom and P is a finite set of productions of the form $\alpha \to \beta$ with $\alpha \in N^*$ and $\beta \in (N \cup T)^*$.

Let $u \Rightarrow v$ denote the derivation relation defined by the application of a production and $\Rightarrow^*$ its reflexive, transitive closure.

The *language* generated by G is the set $L(G) = \{w \in T^* | S \Rightarrow^* w\}$.

Two grammars are equivalent, if they generate the same language.

Grammars are classified by the form of their productions. A grammar $G = (N,T,P,S)$ is *λ-free*, if it has no productions of the form $\alpha \to \lambda$, where λ denotes the empty string. G is *Type 1* or *context-sensitive*, if $|\alpha| \leqslant |\beta|$ for every production $\alpha \to \beta$, *Type 2* or *context-free*, if $\alpha \in N$ and *linear context-free* (*resp. Type 3*), if additionally $\beta \in T^*NT^* \cup T^*$ (resp. $\beta \in T^*N \cup T^*$). A language L is of *Type 0*, if L is generated by a phrase structure grammar. Accordingly, L is called *Type 1*, *Type 2* or *context-free, linear context-free* and *Type 3*.

Recall that the Type 0 languages coincide with the recursively enumerable sets. Type 1 grammars generate the context-sensitive languages, which coincide with the space complexity class NSPACE(n). The Type 2 languages correspond to pushdown automata and the Type 3 languages are the regular sets. These types define the Chomsky hierarchy of languages [19, 20].

In this paper, λ-productions shall play an important role. It is known that λ-productions can be eliminated from arbitrary and from context-free grammars. Up to the empty string λ, which may be attached by a separate production $S \to \lambda$, every recursively enumerable set can be generated by a λ-free grammar and every context-free language can be generated by a λ-free context-free grammar. For arbitrary grammars, replace every λ-production $\alpha \to \lambda$ by productions $A\alpha \to A$ and $\alpha A \to A$ for every symbol $A \in N \cup T$, and for context-free grammars contract λ-paths.

For our new complexity measure we must take a closer look at derivations.

Definition 2. Let $G = (N,T,P,S)$ be a grammar. A derivation from Q_1 to Q_t is a sequence of triples $D = [(Q_i, \alpha_i \to \beta_i, p_i) | i = 1,\ldots,t]$. Here $Q_1 \Rightarrow \ldots \Rightarrow Q_t$ is the sequence of derived strings. For $1 \leqslant i < t, \alpha_i \to \beta_i \in P$ is the production used in the ith step and p_i describes the position of the application, such that $p_i = |u_i| + 1$, if $Q_i = u_i\alpha_i v_i$ and $Q_{i+1} = u_i\beta_i v_i$. In the ith derivation step, the symbols from $Q_i(p_i, p_i + |\alpha_i| - 1)$ are replaced, whereas all other symbols are copied. The last production $\alpha_t \to \beta_t$ and

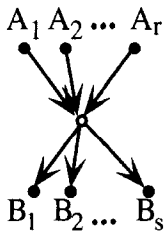

Fig. 1.

p_t are added for convenience of writing. For $i = 1,\ldots,t-2$, the derived string Q_{i+1} has two partitions into substrings according to the application of the ith and $(i+1)$th productions, $Q_{i+1} = u_i\beta_i v_i = u_{i+1}\alpha_{i+1}v_{i+1}$. Accordingly, the $(i+1)$th derivation step occurs to the left of the ith derivation step, if $|u_{i+1}\alpha_{i+1}| \leqslant |u_i|$ and to the right, if $|u_i\beta_i| \leqslant |u_{i+1}|$. Two derivation steps overlap, if $|u_{i+1}| < |u_i\beta_i|$ and $|u_i| < |u_{i+1}\alpha_{i+1}|$. These cases are mutually exclusive and exhaust all possibilities.

A derivation D has a natural representation in terms of a *derivation graph* $\tau(D)$. $\tau(D)$ has two types of vertices, one for the symbols of the derived strings and one for the productions. Every symbol from Q_i is represented by a full vertex "•". These vertices are organized into t layers, and are labelled from left to right by the symbols of the Q_i. Every production $\alpha_i \to \beta_i$ is represented by a subgraph as shown in Fig. 1. It has a vertex "○" for the production, which is placed between the layers for Q_i and Q_{i+1}. If $Q_i = u_i\alpha_i v_i$ and $Q_{i+1} = u_i\beta_i v_i$ then the vertices representing the symbols from u_i and v_i in Q_i and Q_{i+1} are connected by directed edges, which are drawn dashed. There is a full edge from every vertex representing a symbol from α_i to the circle vertex for the production and from there to every vertex from β_i.

In graph theoretic terms, a derivation graph $\tau(D)$ is a directed, acyclic, planar graph with labelled vertices. Its sources are the symbols from the initial string Q_1, its sinks are the symbols from the final string Q_t and the vertices from λ-productions. Some of these sinks from λ-productions are interior vertices. Those on the outer face and the symbols from Q_t are the leaves of $\tau(D)$. These are the real result of D.

Derivation graphs are somewhat redundant. More common are syntactical graphs [21, 22] and derivation trees [19, 20]. These are more compact and are obtained from our derivation graphs by shrinking paths along dashed edges. However, syntactical graphs and derivation trees do not represent a single derivation but a class of derivations with the leftmost derivation as a unique representative.

We now come to the main notion of this paper.

Definition 3. Let $G = (N,T,P,S)$ be a grammar and let $D = [(Q_i, \alpha_i \to \beta_i, p_i) | i = 1,\ldots,t]$ be a derivation from Q_1 to Q_t. For every nonempty substring $y = Q_k(l,r)$ of some Q_k and all previous $Q_i, 1 \leqslant i \leqslant k$, the string of ancestors of y in Q_i is the

non-empty substring of Q_i, defined as follows. For $i = k$ let $anc_k(k,l,r) = Q_k(l_k, r_k)$ with $l_k = l$ and $r_k = r$.
For $i < k$ let $anc_i(k,l,r) = Q_i(l_i, r_i)$, where

$$l_i = \begin{cases} l_{i+1} & \text{if } l_{i+1} < p_i, \\ l_{i+1} - |\beta_i| + |\alpha_i| & \text{if } p_i + |\beta_i| \leqslant l_{i+1}, \\ p_i & \text{if } p_i \leqslant l_{i+1} < p_i + |\beta_i| \end{cases}$$

and

$$r_i = \begin{cases} r_{i+1} & \text{if } r_{i+1} < p_i, \\ r_{i+1} - |\beta_i| + |\alpha_i| & \text{if } p_i + |\beta_i| \leqslant r_{i+1}, \\ p_i + |\alpha_i| & \text{if } p_i \leqslant r_{i+1} < p_i + |\beta_i|. \end{cases}$$

Hence, $anc_i(k,l,r) = anc_{i+1}(k,l,r)$, if the i-th derivation step does not overlap with $anc_{i+1}(k,l,r)$. Otherwise, if $\alpha_i \to \beta_i$ is the actual production, then $anc_i(k,l,r)$ is obtained from $anc_{i+1}(k,l,r)$ by replacing β_i or the portion of β_i contained in $anc_{i+1}(k,l,r)$ by α_i.

The strings of ancestors fully describe the history of a substring y. y only depends on its strings of ancestors and can be retrieved from them by applying the productions of D overlapping with the strings of ancestors. Then y is rewritten in the context of some strings γ and δ, i.e. $Q_i(l_i, r_i) \Rightarrow^* \gamma y \delta$. This property is well known for context-free grammars, where it is decidable whether or not a grammar generates a substring y.

The ancestor width measures the complexity of derivations and strings by the lengths of the strings of ancestors of the productions or equivalently of the leaves of D. The symbols from the derived string do not suffice. Also, the occurrences of the empty string in λ-productions must be taken into account. Our results will make this clear.

Definition 4. Let $G = (N,T,P,S)$ be a grammar. The ancestor width of a derivation $D = [(Q_i, \alpha_i \to \beta_i, p_i) | i = 1, \ldots, t]$ is defined by

$$AW(D) = \max\{|anc_i(k, p_k, p_k + |\alpha_k| - 1)| \mid 1 \leqslant k \leqslant t-1 \text{ and } 1 \leqslant i \leqslant k\}.$$

For a terminal string w let

$$AW(w) = \min\{AW(D) | D : S \Rightarrow^* w \text{ is a derivation of } w\}.$$

Thus, the ancestor width of a string w is taken over all derivations $S \Rightarrow^* w$ and the best one is chosen. This is important. The ancestor width may vary, if the same productions are used in a different order. For example, if there are productions $S \to SB, S' \to B'S'$ and $BXB' \to X$, then applying each production n times yields the derivation $SXS' \Rightarrow^* SXS'$. If the productions are applied n times in the given order, then the strings of ancestors of the occurring X are BXB', SXB', SXS', BXS' and X. However, if the productions $BXB' \to X$ are applied last, then the last X has $B^nXB'^n$ as a string of ancestors.

Notice that we would obtain completely different results, if the ancestor width of a string were defined by the maximal ancestor width of all derivations. This follows from results in [9, 10].

Definition 5. Let $f : \mathbb{N} \to \mathbb{N}$ be a total recursive function. A grammar G is $f(n)$ ancestor width bounded, if $AW(w) \leqslant f(n)$ for every $w \in L(G)$ with $|w| = n$. A language L is $f(n)$ ancestor width bounded, if $L = L(G)$ for some $f(n)$ ancestor width bounded grammar G. Similarly, L is λ-free (resp. context-sensitive) $f(n)$ ancestor width bounded, if there is a corresponding λ-free (resp. context-sensitive) grammar generating L.

3. Ancestor width of grammars

The ancestor width of derivations and strings has a local flavour. It is a punctual measure in the sense that it considers the strings of ancestors of single productions. In this respect it resembles complexity measures for Turing machines, such as crossings, visits, returns or dual returns, see [18, 25] for definitions. On the other hand, time and space are global complexity measures. They take computations as a whole and do not consider details of the internal behaviour of computations and derivations.

From the graphical representation of derivations it is clear that the ancestor width is determined by the leaves of the derivation graph. For a derivation $D = [(Q_i, \alpha_i \to \beta_i, p_i) | i = 1, \ldots, t]$ from Q_1 to Q_t, the leaves of D are the symbols of the derived string, i.e. $Q_t(k,k)$ with $1 \leqslant k \leqslant |Q_t|$, and those distinguished rewritings of the empty string $a_k \to \lambda$, which are not covered by other productions.

Let the *ancestor width of the leaves* be

$$AWL(D) = \max\{\max\{|anc_i(t,k,k)| \mid 1 \leqslant k \leqslant |Q_t|, 1 \leqslant i \leqslant t\} \cup AW(D,\lambda)\},$$

where $AW(D,\lambda) = \max\{|anc_i(k, p_k, p_k + |\alpha_k| - 1)| \mid 1 \leqslant i \leqslant k, 1 \leqslant k \leqslant t, \beta_k = \lambda$, and there is no $r > k$ such that $anc_{k+1}(r, p_r, p_r + |\alpha_r| - 1) = Q_{i+1}(l,r)$ with $l < p_k$ and $r > p_k\}$ is the maximal length of a string of ancestors of a derived occurrences of the empty string λ.

Then we obtain.

Lemma 6. *For every derivation D, $AW(D) = AWL(D)$.*

Consider the ancestor width of context-free grammars and languages. The derivation graphs of context-free grammars are trees. Thus the ancestors of the leaves are singletons and the ancestor width is one. Conversely, if a non-context-free production is applied in a derivation, then the ancestor width is greater than one. Hence, the context-free grammars and languages are the ones with ancestor width one. However, for a grammar G it is undecidable, whether or not G has an ancestor width greater than one. To see this, compose G of two grammars $G = G_1 \cup G_2$, such that $L(G_1) = \Sigma^*$

and every derivation of G_1 has ancestor width $k > 1$ and G_2 is (linear) context-free. Then G has ancestor width one if and only if $L(G_2) = \Sigma^*$, which is an undecidable problem, see e.g. [20].

Next, consider the ancestor width two. Strings of ancestors of length two cannot interact and pass information. In the derivation graphs of such derivations the strings of ancestors surround a face of the underlying planar graph. Now non-context-free productions can be split and simulated by context-free productions. Hence, grammars with ancestor width two generate only context-free languages.

Theorem 7. *If G is a grammar with ancestor width $f(n)$ and $f(n) \leqslant 2$, then $L(G)$ is a context-free language.*

Proof. Let $G = (N, T, P, S)$ and suppose that the productions are of the form $A \to B$, $A \to BC, A \to a, A \to \lambda$ and $AB \to C$, where $A, B, C \in N$ and $a \in T$. Non-context-free productions $\alpha \to \beta$ with $|\alpha| \geqslant 3$ cannot be used and other productions can directly be simulated by such productions with the same ancestor width.

If a non-context-free production $XY \to Z$ is used in some derivation D with $AW(D) = 2$, then XY has a special history. There are an initial production $A_0 \to X_1 Y_1$ and two chains of nonterminal symbols from X_1 to X and from Y_1 to Y, which are obtained by productions of the form $X_i \to u_i X_{i+1}$ and $X_i \to X_{i+1}$ and accordingly $Y_i \to Y_{i+1} v_i$ and $Y_i \to Y_{i+1}$. The strings of ancestors of XY are of the form $X_i Y_j$. Neither non-context-free productions of the form $AB \to C$ nor λ-productions $A \to \lambda$ can be used in these subderivations. They would increase the ancestor width at least to three. The subderivation $A_0 \Rightarrow X_1 Y_1 \Rightarrow^* uXYv \Rightarrow uZv$ with strings u and v is split into $A_0 \Rightarrow X_1 Y_1, X_1 \Rightarrow^* uX \Rightarrow uZ$ and $Y_1 \Rightarrow^* Yv \Rightarrow v$ using only context-free productions. The correctness of this procedure is controlled by additional components at the nonterminals recording the left-hand sides.

For every left-hand side XY of a non-context-free production $XY \to Z$ define their left and right predecessors by

$\text{pre}_l(X) = \{X' \in N |$ there is a rightmost derivation by context-free productions of G of the form $A \to BC$ and $A \to B$ from X' to uX for some $u \in N^*\}$.

$\text{pre}_r(Y) = \{Y' \in N |$ there is a leftmost derivation by context-free productions of G of the form $A \to BC$ and $A \to B$ from Y' to Yv for some $v \in N^*\}$. Let $\text{init}(XY) = \{A_0 \to X_1 Y_1 \in P | X_1 \in pre_l(X)$ and $Y_1 \in pre_r(Y)\}$ be the set of initial productions for $XY \to Z$.

These sets can be effectively computed, since only context-free productions are involved.

Construct the context-free grammar $G' = (N', T, P', S)$ as follows:

$N' = N \cup \{A, XY), (XY, A), (XY, A, UV) | A, X, Y, U, V \in N$ and XY and UV are left-hand sides of non-context-free productions$\}$.

The set P' contains the set of context-free productions of P. Additionally, for every left-hand side XY of a non-context-free production let P' contain the following productions:

for every $A \to X_1Y_1 \in \text{init}(XY)$ and all $A,(A,UV),(UV,A),(U_1V_1,A,U_2V_2) \in N'$ let P' contain the productions

$$A \to (X_1,XY)(XY,Y_1),(A,UV) \to (X_1,XY)(XY,Y_1,UV),$$

$$(UV,A) \to (UV,X_1,XY)(XY,Y_1), \text{ and}$$

$$(U_1V_1,A,U_2V_2) \to (U_1V_1,X_1,XY)(XY,Y_1,U_2V_2).$$

For every production $A \to B \in P$ and all $(A,XY),(XY,A),(XY,A,UV) \in N'$ let P contain the productions

$$(A,XY) \to (B,XY),$$

$$(XY,A) \to (XY,B) \text{ and}$$

$$(XY,A,UV) \to (XY,B,UV).$$

For every production $A \to BC \in P$ and all $(A,XY),(XY,A),(XY,A,UV) \in N'$ let P' contain the productions

$$(A,XY) \to B(C,XY),$$

$$(XY,A) \to (XY,B)C \text{ and}$$

$$(XY,A,UV) \to (XY,B)(C,UV).$$

Finally, for every non-context-free production $XY \to Z$ let P' contain

$$(X,XY) \to Z \text{ and } (XY,Y) \to \lambda.$$

These productions can be effectively computed.

It remains to show that G and G' are equivalent.

Let $w \in L(G)$. If $AW(w) = 1$, then there is a derivation with only context-free productions. Hence $w \in L(G')$, since G' contains all context-free productions of G.

Let $AW(w) = 2$ and let $D = [(Q_i,\alpha_i \to \beta_i, p_i) | i = 1,\ldots,t]$ be a derivation from S to w. Let $XY \to Z$ be a non-context-free production applied at Q_k. Consider the string of ancestors of XY in $Q_{k-1},Q_{k-2},\ldots,Q_j$ such that these strings are of length two for $k-1,\ldots,j+1$ and of length one for j. Let these strings be $X_{k-1}Y_{k-1},\ldots,X_{j+1}Y_{j+1}$ and A_j respectively. From the form of the productions of G, for $i = j+1,\ldots,k$, either $X_i = X_{i+1}$, i.e. X_i is copied in the i-th step, or $X_i \to X_{i+1}$ or $X_i \to X'X_{i+1}$ for some nonterminal X'. By symmetry, either $Y_i = Y_{i+1}$ or $Y_i \to Y_{i+1}$ or $Y_i \to Y_{i+1}Y'$ for some nonterminal Y'. Moreover there is a production $A_j \to X_{j+1}Y_{j+1}$. Any other case, i.e. the application of a λ-production or of another non-context-free production yields an ancestor string of length three. The subderivation on the strings of ancestor of XY between the j-th and k-th steps is simulated by new productions of P', which store XY in the additional components of the nonterminals. Nonterminals of the form (XY,A,UV) are used, if a non-context-free production $UV \to W$ occurs to the right of $XY \to Z$, and A is the common ancestor of Y and U. By induction on the number of non-context-free productions used it follows that $w \in L(G')$.

Conversely, let $D' = [(Q_i, \alpha_i \to \beta_i, p_i) | i = 1, \ldots, t]$ be a derivation from S to some terminal string w in G'. Let j be minimal such that in the jth step new nonterminals are introduced, say by a production $A \to (X_1, XY)(XY, Y_1)$. The XY components disappear only by productions of the form $(X, XY) \to Z$ and $(XY, Y) \to \lambda$. They are passed or are distributed over two nonterminals by productions of the form $(XY, A, UV) \to (XY, B)(C, UV)$.

Now rearrange the derivation D' such that for every XY the erasing of the XY components is done in two successive steps. Notice that a right XY-component disappears only by a production of the form $(X, XY) \to Z$ and the left XY-component diappears only by $(XY, Y) \to \lambda$, where $XY \to Z$ is a production in P. Merge these rewritings into a single step and let $D'' = [(Q_i'', \alpha_i \to \beta_i, p_i) | i = 1, \ldots, t'']$ be the resulting derivation. Then XY-components are adjacent. I.e., if $Q_i'' = uABv$ for some symbols A, B and $A = (A', XY)$, then $B = (XY, B')$. This follows directly from the form of the productions of G'. Let $h : (N' \cup T)^* \to (N \cup T)^*$ be a homomorphism, which erases the components from the new nonterminals. Then $D'' = [(h(Q_i''), h(\alpha_i) \to h(\beta_i), p_i) | i = 1, \ldots, t'']$ is a derivation from S to w in G, where the merged rewritings are non-context-free productions of the form $XY \to Z$. This follows by induction on the length of the derivation.

Moreover, the ancestor width of D'' is bounded by two. To see this consider the strings of ancestors of a left-hand side XY of length greater than one in D''. These are of length two, since in D'' the nonterminals with the XY component are always adjacent, and these are the strings of ancestors. Hence, every derivation D' in G' from S to some terminal string is associated with a derivation of ancestor width at most two from S to w in G. □

This result comes from the fact that non-context-free productions cannot interact in derivations with ancestor width two. A non-context-free production cannot be applied to the string of ancestors of length two, i.e. to the predecessors of another non-context-free production. The derivation graphs of such derivations are "almost" trees and in the planar drawings of these graphs there is no interior face enclosed by another interior face, i.e. the derivation graphs are outerplanar graphs.

To the contrary, ancestor width three is no restriction for the generation of recursively enumerable sets. Thus, we have a jump in complexity from two to three, as it can be observed for many NP-hard problems. First we give a construction with ancestor width four, which is then refined to three.

Theorem 8. *For every recursively enumerable set L there is a grammar G such that $L = L(G)$ and the ancestor width of G is bounded by four.*

Proof. Let $L = L(M)$ for some (nondeterministic) single tape Turing machine M. Assume that M reserves the workspace needed for the computation in a preprocessing phase. Then it begins its real computation, which is described by a sequence of instantaneous descriptions (ID) $K_0 \vdash K_1 \vdash \ldots \vdash K_t$. Every ID is a string of the form $¢uqv\$$,

where q is the state of M, u and v are the contents of the worktape to the left and right of the read-write head and $\not\vdash$ and \$ are the left and right endmarkers, see. e.g. [20]. All IDs have the same length. Successive IDs coincide up to a substring of length two or three around the state q, which describes the used instruction. The set of successive IDs can be described by strings of the form $K_i K'^R_{i+1}$, where K'^R_{i+1} is the primed and reversed copy of K_{i+1}. This set is a (linear) context-free language generated by some grammar $G_1 = (N_1, T_1, P_1, Y)$. Let $G_0 = (N_0, T_0, P_0, X)$ be a (linear) context-free grammar simulating the preprocessing phase of M and generating strings of the form $w\$K'^R_0$. w is a terminal string, whereas all other strings are over the nonterminals of the grammar G.

Let $G_2 = (N_2, T_3, P_2, Z)$ be a (Type 3) context-free grammar generating strings K_t of the form of accepting IDs .

Finally, let $G_3 = (N_3, T_3, P_3, S)$ be a (Type 3) context-free grammar generating strings of the form XY^*Z. The alphabets N_i are pairwise disjoint. All symbols except the ones from the alphabet $T \subseteq T_0$ of the grammar G_0 are nonterminals for G. Every such nonterminal A has a unique primed copy A'. For every such pair add the non-context-free production $A'A \to \lambda$. Let $G = (N, T, P, S)$ be the collection of all these components.

Then $L(G) = L(M)$. To see this, observe that every accepting computation of the Turing machine M on some input string $w \in T^*$ of the form $K_0 \vdash K_1 \vdash \ldots \vdash K_t$ is simulated by a derivation $S \Rightarrow^* XY^{t-1}Z \Rightarrow^* wK'^R_0 K_0 K'^R_1 \ldots K'^R_t K_t \Rightarrow^* w$, where every pair $K'^R_i K_i$ cancels to the empty string λ. Conversely, if G generates a terminal string w, then it must erase all nonterminals to the right of the nonterminal X. This can be achieved only by productions of the form $A'A \to \lambda$, which induces pairs of the form $K'^R K$. Each such K describes an ID of M. Then the derivation can be rearranged into the form as shown in Fig. 2. Every such derivation represents an accepting computation of M on w.

Every derivation D of a string w of the form of Fig. 2 has ancestor width four. D operates "level by level". From left to right it produces the i-th symbol of each ID and then immediately cancels pairs $A'A$. The strings of ancestors of these left-hand sides are of length at most four. They are of the form $XA'AY$ or $YA'AY$ or $YA'AZ$. Notice that the strings of ancestors of the symbols of the generated string w are X and S and are singletons of length one. □

As a consequence we obtain Savitch's normal form for Type 0 grammars [23]. This normal form has been improved by Geffert [17] using only a fixed set of non-context-free productions, e.g. $ABC \to \lambda$.

Corollary 9. *Every recursively enumerable set L can be generated by a grammar which has only context-free productions and productions of the form $AB \to \lambda$.*

For the reduction of the ancestor width to three we modify the derivations and avoid strings of ancestors of the form $YA'AY$. The matching of pairs of symbols $A'A$

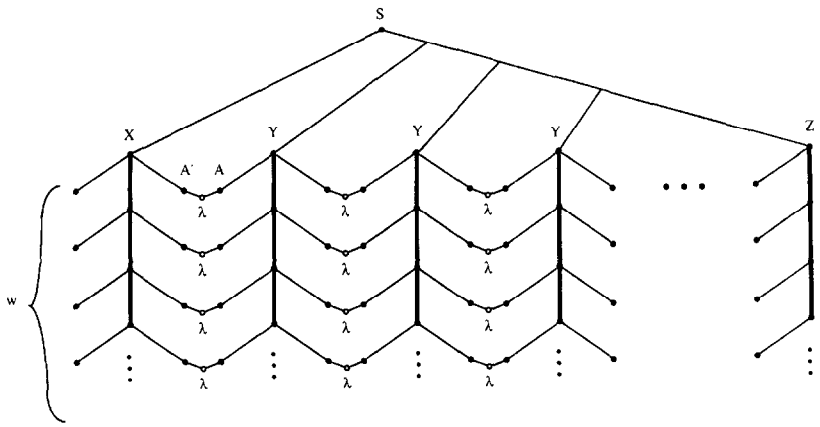

Fig. 2.

is controlled by a distribution. This enforces an exponential expansion of the size of accepting derivation. See Figs. 3 and 4 for an illustration.

Theorem 10. *For every recursively enumerable set L there is a grammar G such that $L = L(G)$ and the ancestor width of G is bounded by three.*

Proof. Consider a Turing machine M as above and use the grammars G_0, G_1 and G_2. Modify G_3 to generate strings of the form $X(IY)^*IZ$, where I is a new nonterminal. Let $\Sigma \cup \{\$\}$ be the set of symbols used for the IDs of M and let Σ' be the set of primed copies. The cancellation rules $A'A \to \lambda$ with $A \neq \$$ from above are now replaced by the following productions

$A'A \to BIB'$ for every $A, B \in \Sigma$,

$I \to AIA'$ for every $A \in \Sigma$,

$A'A \to \$\$$ for every $A \in \Sigma$,

$I \to \$\$'$and $\$'\$ \to \lambda$. Again, A' is the unique primed copy of A.

As above $L(M) = L(G)$. G simulates accepting computations of M by derivations as indicated in Figs 3 and 4. Conversely, every derivation of a terminal string w can be rearranged into a derivation of that form.

Consider a substring YIY and its derivation towards λ. For clarity call them Y_1IY_2. In a leftmost derivation the left spine generates $Y_1 \Rightarrow^* x_0\$\$'x_1'^R$ where $x_0\$$ and $x_1\$$ represent a pair of successive IDs of M. Similarly, the right spine generates $Y_2 \Rightarrow^* x_2\$\$'x_3'^R$.

Consider the string $\$'x_1'^R Ix_2\$$. This string can be rewritten into a string of the form $(\$'\$)^*$ if and only if $x_1^R = x_2$. This is due to the alternation of primed and nonprimed symbols. If $x_2 = A_1A_2 \ldots A_m$ with symbols $A_1, \ldots, A_m$, then

$$\$'A_m' \ldots A_1'IA_1 \ldots A_m\$$$

$$\Rightarrow \$'A_m' \ldots A_1'A_1IA_1'A_1 \ldots A_m\$$$

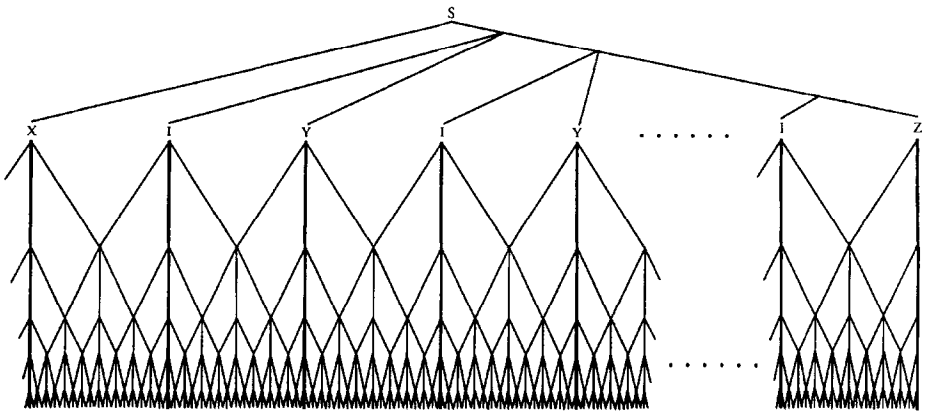

Fig. 3.

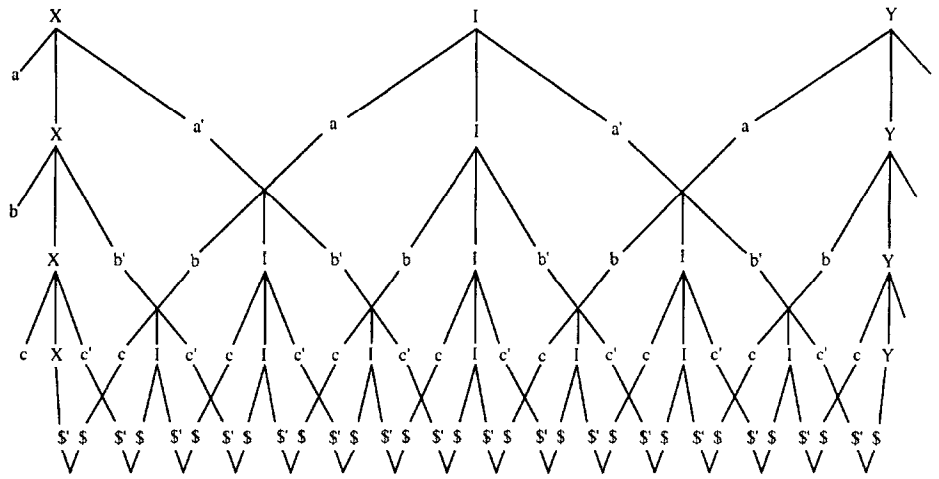

Fig. 4.

$\Rightarrow^* \$'A'_m \ldots A_2 I A'_2 A_2 I A'_2 A_2 I A'_2 \ldots A_m \$$

$\Rightarrow^* \$'A'_m (A_m I A'_m)^{2^m-1} A_m \$$

$\Rightarrow^* (\$'\$)^{2^{m+1}}$

$\Rightarrow^* \lambda.$

Consider the strings of ancestors of $'$ in derivations as in Fig. 3. They are of the form $'$, $UA'A, A'AU, UAU$ and $A'U$, where U stands for I, X, Y or Z. The ancestor width of such derivations is three, and every string can be generated in such a way. □

The ability to generate arbitrary recursively enumerable sets with low ancestor complexity comes from the extensive use of λ-productions and the transfer of the "real" computation to the ancestors of occurrences of λ. Hence, only two classes of languages remain, the context-free languages with ancestor width one and two, and the recursively enumerable sets with ancestor width three. Thus, there is no recursive relationship to other complexity measures, and the ancestor width is not a complexity measure satisfying Blum's axioms [20, 25].

4. Ancestor width of λ-free grammars

In λ-free grammars the symbols from the derived string are the only leaves of a derivation graph. Hence, the ancestor width and the workspace of derivations and strings differ at most by the factor n where n is the length of the derived string. The workspace of a derivation is the maximal length of the occurring strings. The workspace of grammars is one-to-one related to the space complexity of nondeterministic single tape Turing machines.

Let NSPACE(f) denote the complexity class defined by nondeterministic $f(n)$ space bounded Turing machines. For $f(n) \geqslant n$, NSPACE(f) is the class of languages generated by grammars with workspace bounded by $f(n)$. Let $\mathrm{AW}_\lambda(f)$ ($\mathrm{AW}_{cs}(f)$) denote the class of languages generated by λ-free (contest-sensitive) grammars with ancestor width bounded by $f(n)$ and denote the class of context-free languages by CFL. Then we obtain:

Theorem 11. *For every function f*

$$CFL \subseteq AW_{cs}(f) \subseteq AW_\lambda(f) \subseteq NSPACE(n \cdot f(n))$$

and for $f(n) \geqslant n$

$$NSPACE(f) \subseteq AW_\lambda(f) \subseteq NSPACE(n \cdot f(n)).$$

In particular, if $f(n) \leqslant k$ for some k, then λ-free grammars with ancestor width $f(n)$ can generate only context-sensitive languages. Many well-known languages, such as $\{wcwc | w \in \{a,b\}^*\}$, $\{a^n b^n c^n | n \geqslant 1\}$ and $\{(a^n c)^n | n \geqslant 1\}$ can be generated by such grammars with ancestor width four. Moreover, any language L can be generated by a context-sensitive or λ-free grammar with small ancestor width, if L is represented with sufficiently long tails. Clearly, every context-sensitive language can be generated by a context-sensitive grammar with linear ancestor width.

Example 12. The language $\{wcwc | w \in \{a,b\}^*\}$ can be generated by a λ-free grammar with ancestor width four.

Proof. Consider a grammar as in Theorem 2. All symbols except a, b and c are nonterminals. Let, $\bar{a}$, a' and a'' be distinct copies of the symbol a, and similarly for b.

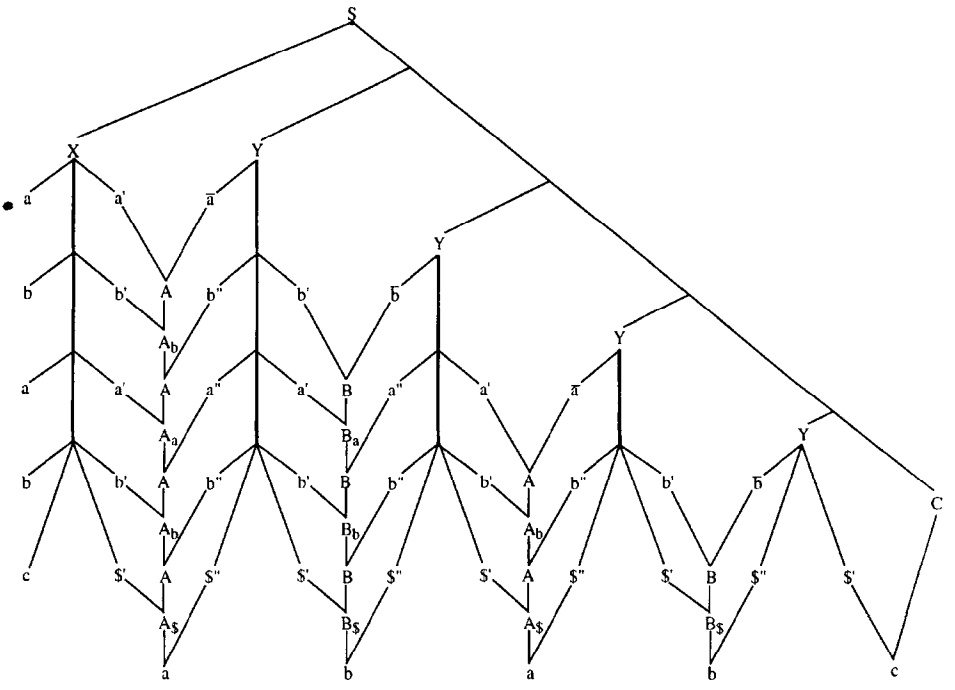

Fig. 5.

The underlying idea is similar to the derivation shown in Fig. 2, where instead of λ's we generate the second copy of wc. Fig. 5 illustrates the derivation of $ababcababc$.

Let G first generate strings of the form XY^*C. In a linear context-free fashion X generates strings $wc\$'w'^R$ with $w \in \{a,b\}^*$ and $w'^R \in \{a',b'\}^*$ being the primed and reversed copy of w. Let Y generate strings of the form $du''\$''\$'u'^R$ with $d \in \{\bar{a},\bar{b}\}, u'' \in \{a'',b''\}^*, u' \in a',b'^*$ and u'^R being the primed and reversed copy of u.

Let A,B,A_a,A_b,B_a,B_b be new nonterminals. These symbols remember the barred symbol with A,A_a and A_b for $\bar{a}$ and B,B_a and B_b for $\bar{b}$. The indices handle the matching. Add the non-context-free productions $a'\bar{a} \to A, a'A \to A_a, A_a a'' \to A, b'A \to A_b, A_b b'' \to A, \$'A \to A_\$$ and $A_\$\$'' \to a$ and similarly $b'\bar{b} \to B, a'B \to B_a, B_a a'' \to B, b'B \to B_b, B_b b'' \to B, \$'B \to B_\$$ and $B_\$\$'' \to b$. Finally add $\$'C \to c$. Using these productions strings of the form XY^nC can be transformed into strings of the form $wcwc$ if and only if $n = |w|$ and the ith symbol of the second w is derived from the barred symbol created by the ith Y. Concerning the ancestor width, if the derivations are done level by level, then long strings of ancestors are of the form $Ya'a''Y, Ya'\bar{a}Y, Ya'AY$ or YA_aaY. □

This derivation scheme can be extended to the generation of more complex languages including $\{a^n b^n c^n | n \geq 1\}$.

Example 13. The languages k-COPY $=\{(wc)^k | \{w \in \{a,b\}^*\}, k \geqslant 1\}$ and w-COPY$=$ $\{(wc)^n | \{w \in \{a,b\}^*\}, n = |w|\}$ and similarly languages such as $\{a^n b^n c^n | n \geqslant 1\}$ and $\{(a^n c)^n | n \geqslant 1\}$ can be generated by λ-free grammars with ancestor width four.

Proof. The grammar from Example 1 is extended by some extra book-keeping. For n-COPY with $n = k$ for some k or $n = |w|$ let X generate strings of the form $(wc)^{n-1}$ $\$'(c'w'^R)^{n-1}$, otherwise the derivation does not end with a terminal string.

If n is bounded, then the proper number of c's is counted using new nonterminals. The final wc is generated to the right of X. Clearly, it must be checked that all copies of w coincide. This is done symbol by symbol, such that the ith symbol of each copy of w is checked by the subderivation to the left of the ith Y.

The grammar G first generates strings of the form $X\{Y_a, Y_b\}^n C$. Here, Y_a and Y_b are new nonterminals. $Y_d \in \{Y_a, Y_b\}$ generates strings of the form $du_1'' c'' du_2'' c'' \ldots du_r'' c''$ $\$''\$' c' u_r'^R \ldots c' u_2'^R c' u_1'^R$ with $d \in \{\bar{a}, \bar{b}\}, u_j'' \in \{a'', b''\}^*$, $u_j' \in \{a', b'\}^*$, and $u_j'^R$ being the primed and reversed copy of u_j'' for $0 \leqslant j \leqslant r$. r must decrease by 1 from left to right. Y_d produces extra d's only to the left. This d determines the terminal symbol derived from the subderivation by the productions $a'\bar{a} \to A$, and $b'\bar{b} \to B$, and the capital letters so obtained are passed by the other productions.

The coincidence of the copies of wc is guaranteed by pairs of double and single primed copies of a, b, c and \$, and subderivations of the form $a'Aa'' \Rightarrow^* A$. If the derivations are done level by level, then their ancestor width is bounded by four. Languages such as $\{a^n b^n c^n | n \geqslant 1\}$ and $\{(a^n c)^n | n \geqslant 1\}$ can be obtained by a renaming of symbols. □

The derivation schemes from Theorems 2 and 3 are now applied to a represention of arbitrary languages with "tails". This padding technique is used in Translational Lemmas in complexity theory and has been investigated by Book [5]. Strings from a given language are expanded by tails, i.e. w is transformed into wd^t, where d is a new symbol. So a language with a high complexity is transformed into one with a lower complexity. In the Translational Lemmas for complexity classes [5] the lengths of the tails measure the difference between the bounding functions for the complexity classes. Thus, the length of the tails can be used as a complexity measure. In our case, if a λ-free grammar with ancestor width four generates languages from NP, then it has tails of polynomial length. However, with ancestor width three the tails are exponential in the size of the space used and polynomial in the time used. Thus, in the terminology of [7], every language in NP has a polynomial representative generated by a λ-free grammar with ancestor width four, and every language in PSPACE has an exponential representative generated by a context-sensitive grammar with ancestor width three.

Theorem 14. *Let L be a language accepted by a nondeterministic Turing machine M running in time $T(n)$ and using space $S(n)$ with $T(n) \geqslant S(n) \geqslant n$ and T and S constructible.*

(1) *There is a λ-free grammar G with ancestor width four such that $L(G)$ represents L with tails of length $T(n)$.*
$L(G) = \{wd^t | w \in L(M) \text{ and } t = T(|w|)\}$.

(2) *There is a context-sensitive grammar G' with ancestor width three such that $L(G')$ represents L with tails of length $T(n) \cdot 2^{S(n)}$.*
$L(G') = \{wd^t | w \in L(M) \text{ and } t = T(|w|) \cdot 2^{S(|w|)}\}$.

Proof. (1) Consider the grammar G constructed in Theorem 2. Modify G to a λ-free grammar G and replace the λ-productions of the form $A'A \to \lambda$ by productions $A'I \to I_A, I_A A \to \lambda$ and $I \to d$, where d is a new terminal symbol and I and I_A are new nonterminal symbols. As in Theorem 3, let the subgrammar G_3 generate strings of the form $XI(YI)^*Z$. Then the matching of pairs of symbols $A'A$ is done by derivation steps $YIY \Rightarrow AYA'IY \Rightarrow AYI_AY \Rightarrow AYI_AAYA' \Rightarrow AYIYA'$. These are applied level by level to guarantee strings of ancestors of length at most four, e.g. $YA'IY$ or YI_AAY.

(2) Replace the only erasing production $\$'\$ \to \lambda$ from the proof of Theorem 3 by $\$'\$ \to dd$. Suppose that a computation of M of length t and space s is described by $t-1$ instantaneous descriptions, each of length $s-2$. Then the grammar G' generates strings wd^r where $r = t \cdot 2^s$. □

Theorem 5 reveals that the languages generated by λ-free or context-sensitive grammars with ancestor width $k \geqslant 3$ have a context-sensitive behaviour. The classes $\mathrm{AW}_\lambda(k)$ and $\mathrm{AW}_{CS}(k)$ are close to arbitrary context-sensitive languages. For example, they have the same decision properties. The membership problem, i.e. $w \in L(G)$, is decidable, whereas, the emptiness problem, i.e. $L(G) = \emptyset$, is undecidable.

Regarding closure properties, it is readily seen that the classes $\mathrm{AW}_\lambda(k)$ and $\mathrm{AW}_{CS}(k)$ are closed under union, concatenation, star, nonerasing homomorphism and intersection with regular sets. This can be shown by the usual constructions with grammars. We do not know whether or not these classes are closed under inverse homomorphism. The grammar based proofs of a closure under inverse homomorphism apply a compression technique, as it is used in speed-up theorems. However, speed-up techniques don't work for grammars with ancestor width. This shows the jump from the ancestor widths two to three. Thus it is unsolved whether or not there is a strict hierarchy of classes of languages with bounded ancestor width, i.e. whether the inclusion $\mathrm{AW}_\lambda(k) \subseteq \mathrm{AW}_\lambda(k+1)$ is proper for $k \geqslant 3$. However, we feel that these questions are not of a primary concern.

5. Conclusion

Our results show that the notion of ancestor width used in this paper does not fully explain the way in which grammars generate non-context-free languages. However, one may vary the notion of ancestor width and restrict oneself to leftmost derivations or consider the maximum – and not the minimum – over all derivations. Then

the ancestor complexity is closely related to the space bounds of one-way auxiliary pushdown automata [8] and to Chytil's approaches towards the context-sensitivity of languages [13, 14, 16]. However, Book's question on the use of context in context-sensitive derivations is still unanswered.

References

[1] B.S. Baker, Non-context-free grammars generating context-free languages, Inform. and Control 24 (1974) 231–243.

[2] R.V. Book, Grammars with time functions, Mathematical Lingustics and Automatic Translation, NSF 23, Harvard University, Cambridge, MA, 1969.

[3] R.V. Book, Time-bounded grammars and their languages, J. Comput. System Sci. 5 (1971) 397–418.

[4] R.V. Book, Terminal context in context-sensitive grammars, SIAM J. Comput. 1 (1972) 20–30.

[5] R.V. Book, On the structure of context-sensitive grammars, Internat. J. Comput. Inform. Sci. 2 (1973) 129–139.

[6] R.V. Book, Translational lemmas, polynomial time and $(\log n)^j$-space, Theoret. Comput. Sci. 1 (1976) 215–226.

[7] R.V. Book, On the complexity of formal grammars, Acta Inform. 9 (1978) 171–181.

[8] F.J. Brandenburg, On one-way auxiliary pushdown automata, Lecture Notes in Computer Science, vol.48 Springer, Berlin, 1977, pp. 132–144.

[9] F.J. Brandenburg, The contextsensitivity bounds of contextsensitive grammars and languages, Lecture Notes in Computer Science, vol.52, Springer, Berlin, 1977, pp. 120–132.

[10] F.J. Brandenburg, Die Zusammenhangskomplexität von nicht-kontext-freien Grammatiken, Dissertation, Universität Bonn, 1978.

[11] F.J. Brandenburg, On the height of synactical graphs, Lecture Notes in Computer Science, vol. 104 Springer, Berlin, 1981, pp. 13–21.

[12] F.J. Brandenburg, On the transformation of derivation graphs to derivation trees, Lecture Notes in Computer Science, vol. 118, Springer, Berlin, 1981, pp. 224–233.

[13] M.P. Chytil, Analysis of the non-context-free component of formal languages, Lecture Notes in Computer Science, vol. 45, Springer, Berlin, 1976, pp. 230–236.

[14] M.P. Chytil, Charaterization of context-sensitivity by grammars and automata, Twente University of Technology, Netherlands, Memorandum no. INF. 82–9, 1982.

[15] M.P. Chytil, Almost context-free languages, Fund. Inform. IX (1986) 283–322.

[16] M.P. Chytil, Kins of context-free languages, Lecture Notes in Computer Science, vol. 233, Springer, Berlin, 1986, pp. 45–58.

[17] V. Geffert, Grammars with context dependency restricted to synchroniztion, Lecture Notes in Computer Science, vol. 233, Springer, Berlin, 1986, pp. 370–378.

[18] S.A. Greibach, Visits, crosses and reversals for nondeterministic offline machines, Inform. and Control 36 (1978) 174–216.

[19] M.A. Harrison, Introduction to Formal Language Theory, Addison-Wesley, Reading, MA, 1978.

[20] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, MA, 1979.

[21] T. Kamimura, G. Slutzki, Transductions an dags and trees, Math. Systems Theory 15 (1982) 225–249.

[22] J. Loeckx, The parsing of general phase-structure grammars, Inform. and Control 6 (1970) 443–464.

[23] W.J. Savitch, How to make arbitrary grammars look like context-free grammars, SIAM J. Comput. 2 (1973) 174–182.

[24] K.W. Wagner, Do there exist languages with an arbitrarily small amount of context-sensitivity, Lecture Notes in Computer Science, vol. 270, Springer, Berlin, 1987, pp. 427–432.

[25] K.W. Wagner, G. Wechsung, Computational Complexity, VEB Deutscher Verlag der Wissenschaften, Berlin, 1986.