ON THE COVERING OF LEFT RECURSIVE GRAMMARS.

A. NIJHOLT

Free University
Department of Mathematics
Amsterdam, The Netherlands

ABSTRACT.

In this paper we show that some prevailing ideas on the elimination of left recursion in a context-free grammar are not valid. An algorithm and a proof are given to show that every proper context-free grammar is covered by a non-left-recursive grammar.

Keywords: cover, left-recursion, context-free grammar, parsing.

1. INTRODUCTION.

There exists a well-known method for eliminating *left recursion* in a context-free grammar. The motivation for eliminating left recursion is for example that certain parsing algorithms do not work for left-recursive grammars.
However with this usual method the parses of the original grammar cannot, in general, be reconstructed in a simple way from the parses of the non-left-recursive grammar obtained by this method. That is, the new grammar does not *cover* the original grammar.
There has been some research on the covering of grammars by grammars which are in a certain normal form. In general the possibility to cover a grammar depends on the definition of cover which is used. Examples of those definitions can be found in [1,p.276], [2] and [3], and we will discuss them as far as necessary for our purposes in the next section.
In [1] en [2] some remarks can be found from which one could conclude that elimination of left recursion changes the structure of a grammar in such a way that there is no covering grammar. However, in our opinion, and not only in the case of elimination of left recursion, the relation between changes of structure (whatever is meant by structure) and covers is not so close as suggested. This point is also discussed, though rather informally, in the next section. In the case of elimination of left recursion we show in section 3 that this can be done in such a way that the grammar obtained covers the original grammar.

In the remainder of this introduction we give some definitions and notational conventions. In section 2 we discuss the definition of cover and the usual algorithm for eliminating left recursion. In section 3 we give our algorithm for eliminating left recursion, which is just a slight variation of the usual method, and prove its correctness and its covering property. Moreover we give an example of the use of this algorithm and we conclude in section 4 with a result which was inspired by a more practical consideration of the elimination of left recursion.

Preliminaries.
We review some basic concepts concerning formal grammars. This material can also be found in [1].

DEFINITION 1.1. A *context-free-grammar* (cfg for short) is a four-tuple $G = (N,\Sigma,P,S)$, where N is the alphabet of *nonterminals*, Σ is the alphabet of *terminals*, $N \cap \Sigma = \phi$ (the empty set), $N \cup \Sigma = V$, $S \in N$, and the set of *productions* P is a finite subset of $N \times V^*$.

Instead of writing $(A,\alpha) \in P$, we write $A \to \alpha$ in P. Let $u, v \in V^*$, then $u \Rightarrow v$ holds if there exist $x, y, w \in V^*$ and $A \in N$ such that $u = xAy$, $v = xwy$ and $A \to w$ is in P. If $x \in \Sigma^*$ we write $u \underset{\ell}{\Rightarrow} v$ and if $y \in \Sigma^*$ we write $u \underset{r}{\Rightarrow} v$.

The reflexive-transitive closures on V^* of these relations are written as $\overset{*}{\Rightarrow}$, $\overset{*}{\underset{\ell}{\Rightarrow}}$ and $\overset{*}{\underset{r}{\Rightarrow}}$ respectively, while the transitive closures are written as $\overset{+}{\Rightarrow}$, $\overset{+}{\underset{\ell}{\Rightarrow}}$ and $\overset{+}{\underset{r}{\Rightarrow}}$.

The set $L(G) = \{x \in \Sigma^* \mid S \overset{*}{\Rightarrow} x\}$ is the *language* generated by G. If $u_o \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \dots \Rightarrow u_r$, then this sequence is called a *derivation* of u_r from u_o. If instead of $\Rightarrow$ the relation $\underset{\ell}{\Rightarrow}$ or the relation $\underset{r}{\Rightarrow}$ is used, then this sequence is said to be a *leftmost derivation* or a *rightmost derivation* respectively. If in a leftmost or a rightmost derivation of u_r from u_o, for each $0 \leq i \leq r$, u_{i+1} is obtained from u_i by applying production $\Pi_i = A_i \to y_i$ then, in the case of a leftmost derivation the sequence $\Pi_o\Pi_1 \dots \Pi_{r-1}$ is said to be a *left parse* of u_r from u_o, and in the case of a rightmost derivation the sequence $\Pi_{r-1}\Pi_{r-2} \dots \Pi_o$ is said to be a *right parse* of u_r to u_o.

If $u_o = S$ then each u_i, $o \leq i \leq r$, is called a *sentential form*. A cfg G is said to be *ambiguous* if there is $w \in L(G)$ such that w has at least two left parses.

A nonterminal A is said to *left-derive* x, where $x \in V$, if $A \overset{+}{\Rightarrow} x\alpha$ for some $\alpha \in V^*$.

DEFINITION 1.2. Let Σ_1 and Σ_2 be alphabets. A function f from Σ_1 into Σ_2^* is extended to a *homomorphism* from Σ_1^* into Σ_2^* by the conditions $f(\varepsilon) = \varepsilon$ and $f(a_1a_2\dots a_n) = f(a_1)f(a_2)\dots f(a_n)$, where ε is the empty string and a_i, $1 \leq i \leq n$, is in Σ_1. The homomorphism f is called *fine* if, for each $a \in \Sigma_1$, $f(a) \in \Sigma_2 \cup \{\varepsilon\}$.

DEFINITION 1.3. A cfg $G = (N, \Sigma, P, S)$ is said to be *reduced* if each element of V appears in some sentential form and each nonterminal of G can derive a string of terminals. Cfg G is said to be *cycle-free* if there is no derivation of the form $A \overset{+}{\Rightarrow} A$, for any $A \in N$.

G is said to be ε-*free* if there are no productions of the form $A \to \varepsilon$, where $A \neq S$, in P. In the sequel we will only consider grammars which are reduced and cycle-free. A cfg G is said to be *proper* if it is reduced, cycle-free and ε-free.

A nonterminal A is said to be *left-recursive* if $A \overset{+}{\Rightarrow} A\beta$ for some $\beta \in V^*$. A cfg G is said to be left recursive if G has at least one left-recursive nonterminal.

A cfg G is said to be in *Greibach normal form* (GNF) if G has only productions of the form $A \to a\alpha$, where $a \in \Sigma$ and $\alpha \in N^*$, or $S \to \varepsilon$.

2. ELIMINATION OF LEFT RECURSION.

First we give the usual method for eliminating left recursion. Our starting-point is a proper grammar $G = (N, \Sigma, P, S)$, where $N = \{A_1, A_2, \dots, A_n\}$.

ALGORITHM

(1) Set $i = 1$.

(2) Let the A_i-productions be
$A_i \to A_i\alpha_1 | \dots | A_i\alpha_m | \beta_1 | \beta_2 | \dots | \beta_p$
where no β_j, $1 \leq j \leq p$, begins with A_k if $k < i$.
Replace these productions by
$A_i \to \beta_1 | \beta_2 | \dots | \beta_p | \beta_1 C_i | \dots | \beta_p C_i$, and
$C_i \to \alpha_1 | \dots | \alpha_m | \alpha_1 C_i | \dots | \alpha_m C_i$ where
C_i is a new nonterminal.

(3) If $i = n$, then halt. Otherwise, set $i = i + 1$ and $j = 1$.

(4) Replace each production of the form $A_i \to A_j\alpha$ by the productions $A_i \to \beta_1\alpha | \dots | \beta_m\alpha$, where $A_j \to \beta_1 | \dots | \beta_m$ are all the A_j-productions.

(5) If $j = i - 1$ go to step (2). Otherwise set $j = j + 1$ and go to step (4).

In general we want to compare the parses of the original grammar G with the parses of a grammar G' obtained from G by transformation. Therefore we give a definition which can be found in [1]. We assume that the productions of each grammar are numbered for identification.

We identify these numbers with the productions.

DEFINITION 2.1. Let $G = (N, \Sigma, P, S)$ and $G' = (N', \Sigma, P', S')$ be two cfg's such that $L(G) = L(G')$. In the following two conditions x and y are variables with domain {left, right}. Let $w \in L(G')$ and let $h : P'^* \to P^*$ be a homomorphism such that

(i) if π' is an x-parse for w with respect to G' then $h(\pi')$ is a y-parse for w with respect to G, and

(ii) if π is a y-parse for w with respect to G then there exists π' such that $h(\pi') = \pi$ and π' is an x-parse of w with respect to G'.

If in (i) and (ii) both x and y are replaced by 'left', then G' is said to *left-cover* G. If both x and y are replaced by 'right' then we say that G' *right-covers* G.

If x is replaced by 'left' and y is replaced by 'right' then we say that G' *left-to-right-covers* G.

The definiton of (complete) cover given in [2] can shown to be equivalent to the definition of right cover given here if the cover-homomorphism h is fine.

EXAMPLE.

G' with the only production 1. $S' \to ab$ right-covers G with productions 1. $S \to aB$ and 2. $B \to b$. The cover-homomorphism h is defined by $h(1) = 21$.

G' cannot cover G with a cover-homomorphism which is fine. In this paper we will only make use of a fine homomorphism. Therefore the definition of right cover used here and the definition of complete cover in [2] are equivalent. Now we can ask whether it is possible that a grammar G' obtained after eliminating left recursion from a cfg G, covers G. Therefore we consider the following two grammars, G_1 and G' (only the productions are displayed).

G_1 with productions	G' with productions	
1. $S \to So$	1. $S \to o$	5. $C \to o$
2. $S \to S1$	2. $S \to 1$	6. $C \to 1$
3. $S \to o$	3. $S \to oC$	7. $C \to oC$
4. $S \to 1$	4. $S \to 1C$	8. $C \to 1C$

G' is obtained from G_1 by eliminating left recursion according to the ususal algorithm. It can easily be verified that G' neither left-covers nor right-covers G_1. In this case we have G' left-to-right-covers G_1, but that is not true in general which can be seen by eliminating left recursion from the grammar G_2 with the following productions:

1. $S \to Aa$	5. $A \to Ao$
2. $S \to Ab$	6. $A \to A1$
3. $S \to o$	7. $A \to o$
4. $S \to 1$	8. $A \to 1$

Now consider the grammar G_1' with productions

1. $S \to C$	(ε)	5. $D \to o$	(1)
2. $S \to CS'$	(ε)	6. $D \to 1$	(2)
3. $S' \to D$	(ε)	7. $C \to o$	(3)
4. $S' \to DS'$	(ε)	8. $C \to 1$	(4)

In this case we have G_1' right-covers G_1, where the cover-homomorphism h is defined by $h(1) = h(2) = h(3) = h(4) = \varepsilon$ and $h(5) = 1$, $h(6) = 2$, $h(7) = 3$ and $h(8) = 4$, which was already indicated between parentheses after each production displayed above. G_1' is not left-recursive and in the following section we shall show that this is not by accident. Notice that the parse trees of G_1' do not differ very much of the parse trees of G'. The parse trees have the same skeleton. However G' is in Greibach normal form while G_1' is not. This will turn out to be essential.

At this moment it is necessary to look at some remarks in the literature.

First we quote from [2, p.679].

"We would like to say G' covers G if given a parser for G' one can construct a parser for G. The motivation for this is that parsers typically handle grammars in some normal form. Presented with an arbitrary grammar G it may be possible to transform it into a grammar G' which is in this normal form. In what cases can a parser for G' be used to produce a parser for G? For example, simple top-down parsers will not tolerate left-recursive rules which allow $A \overset{+}{\Longrightarrow} Ax$ for some nonterminal A and string x. However, given a grammar G there is a grammar G' equivalent to G which has no such left-recursive rules. Can one construct a parser for G given a parser for G'? We shall prove that the answer is no, given our definition of covering".

However the 'proof' is introduced with the following remark [2, p.686].

"We now embark on the proof of another negative result by exhibiting a grammar which cannot be covered by any grammar in Greibach form.
Thus the elimination of left recursive changes the structure of a grammar sufficiently that it cannot have a covering grammar."

And then a proof is given that cfg G_1 we displayed above cannot be right-covered by a cfg in GNF. To us it is not clear why one can conclude from this that the elimination of left recursion plays such an important role. We can find the same conception in [1] from which we quote (p.283):

"We should observe that the condition of cycle freedom plus no ε-productions is not really very restrictive.
Every context-free language without ε has such a grammar, and moreover, any context-free grammar can be made cycle-free and ε-free by simple transformations (...). What is more, if the original grammar is unambiguous then the modified grammar left and right-covers it. Non-left recursion is a more stringent condition in this sense. While every context-free language has a non-left-recursive grammar, there may be no non-left-recursive covering grammar."

We do not know whether the first part of these remarks (elimination of ε-productions) is correct, however for the second part (elimination of left recursion) in [1] the reader is referred to the cfg G_1 which we already discussed above and which is on the contrary a grammar for which we can find a non-left-recursive covering grammar. In the next section we shall show that this remark is not correct.

3. ON THE COVERING OF LEFT-RECURSIVE GRAMMARS.

In this section we give, and prove the correctness of, an algorithm for the elimination of left recursion in a cfg such that the cfg obtained right-covers the original grammar.

ALGORITHM 3.1. Elimination of left recursion

Input. A proper cfg $G = (N, \Sigma, P, S)$, G is left-recursive.

Output. A non-left-recursive cfg G' which right-covers G.

Method. The following notations are used. Let $N = \{A_1, A_2, \ldots, A_r\}$. The notation $(j)\colon A_n \to \rho\ (k)$, means that the production $j = A_n \to \rho$ is mapped on a production k of grammar G.

Initially we have for each production $(k)\colon A_n \to \rho\ (k)$. This notation, if necessary, is extended to

$(j_1, j_2, \ldots, j_p)\colon A_n \to \rho_1 | \rho_2 | \ldots | \rho_p\ (k_1, k_2, \ldots, k_p)$

or we say that the productions $(j_1, j_2, \ldots, j_p)$ are mapped on the productions $(k_1, k_2, \ldots, k_p)$. (Some of the k_i's, $1 \le i \le p$, may be ε).

(1) Set $i = 1$

(2) Let the A_i productions be
$A_i \to A_i\alpha_1 | A_i\alpha_2 | \ldots | A_i\alpha_m | \beta_1 | \beta_2 | \ldots | \beta_n$
$(i_1, i_2, \ldots, i_m, \ldots, i_{m+n})$ where each β_j, $1 \le j \le n$, begins with a terminal or some A_k such that $k > i$. If $m = o$, go to step (3).
Replace these A_i-productions by

$A_i \to C_i \mid C_iA_1'$ $(\varepsilon, \varepsilon)$

$A_1' \to D_i \mid D_iA_i'$ $(\varepsilon, \varepsilon)$

$D_i \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_m$ $(i_1, i_2, \ldots, i_m)$

$C_i \to \beta_1 \mid \beta_2 \mid \ldots \mid \alpha_n$ $(i_{m+1}, i_{m+2}, \ldots, i_{m+n})$

where C_i, D_i and A_i' are newly introduced nonterminals.

(3) If $i = r$, let G' be the resulting grammar, and halt. Otherwise, set $i = i + 1$ and $j = 1$.

(4) Let $A_i \to A_j\gamma_1 | A_j\gamma_2 | \ldots | A_j\gamma_2$ $(r_1, r_2, \ldots, r_q)$ be all A_i-productions of which the right-hand sides begin with nonterminal A_j. We distinguish between two cases (a) and (b).
(a). <u>C_j is defined</u>.
Suppose we have A_j-productions
$A_j \to C_j | C_jA_j'$ $(\varepsilon, \varepsilon)$
and C_j-productions
$C_j \to X_1\delta_1 | X_2\delta_2 | \ldots | X_p\delta_p$ $(s_1, s_2, \ldots, s_p)$
where X_ℓ, $1 \le 1 \le p$, may be a terminal or a nonterminal.
Replace each production $A_i \to A_j\gamma_k$ (r_k), $1 \le k \le q$, by

$A_i \to X_1H_j^1\gamma_k | X_2H_j^2\gamma_k | \ldots | X_pH_j^p\gamma_k$ $(r_k, r_k, \ldots, r_k)$

and add productions, for $1 \leq \ell \leq p$,

$H_j^\ell \rightarrow Q_j^\ell A_j'$ $\quad(\varepsilon)$

$H_j^\ell \rightarrow Q_j^\ell$ $\quad(\varepsilon)$

$Q_j^\ell \rightarrow \delta_\ell$ $\quad(s_\ell)$

where H_j^ℓ and Q_j^ℓ, $1 \leq \ell \leq p$, are newly introduced nonterminals.

(b). <u>C_j is not defined</u>.

Suppose we have A_j-productions

$A_j \rightarrow X_1\delta_1|X_2\delta_2|\dots|X_p\delta_p \quad (s_1,s_2,\dots,s_p)$

where X_ℓ, $1 \leq \ell \leq p$, may be a terminal or a nonterminal. Replace each production

$A_i \rightarrow A_j\gamma_k \quad (r_k)$, $\quad 1 \leq k \leq q$, by

$A_i \rightarrow X_1H_j^1\gamma_k|X_2H_j^2\gamma_k|\dots|X_pH_j^p\gamma_k \quad (r_k,r_k,\dots,r_k)$

and add productions, for $1 \leq \ell \leq p$,

$H_j^\ell \rightarrow \delta_\ell \quad (s_\ell)$

where H_j^ℓ, $1 \leq \ell \leq p$, is a newly introduced nonterminal.

(5) If $j = i - 1$, go to step (2). Otherwise set $j = j + 1$ and go to step (4).

<u>End of the algorithm</u>.

To prove the correctness of this algorithm we need some additional notations. Let $\alpha \in V^*$ then we have

$L(\alpha)$ is the language generated from α

$s(\alpha)$ is a sentence in $L(\alpha)$

$rs(\alpha)$ is a right parse of $s(\alpha)$ to α

$R(\alpha)$ is the set of right parses of sentences in $L(\alpha)$ to α.

<u>EXAMPLE</u>.

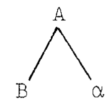
Let G be cfg with productions

1. $S \rightarrow AbC$	3. $C \rightarrow aC$
2. $A \rightarrow a$	4. $C \rightarrow d$

then $R(aC) = \{43^n | n \geq o\}$, $R(AbC) = R(A)R(C)$, and if $s(bC) = ba^nd$ for a certain n, $n \geq o$, then $rs(bC) = 43^n$ for this sentence. Notice that since in general G may be ambiguous (that is, one sentence may have more than one right parse), $rs(\alpha)$ is a set of right parses. However, the proof is such that without loss of generality we may assume that $rs(\alpha)$ is the representation of one of the right parses in $rs(\alpha)$ and therefore we can handle $rs(\alpha)$ as a string.

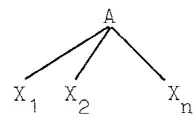
In the parse trees we display we will, if possible, use

or

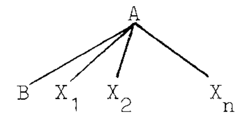


where $\alpha = X_1X_2\dots X_n$,

rather than



or



<u>THEOREM 3.1</u>.

Every proper, left recursive context-free grammar is right-covered by a non-left-recursive context-free grammar.

<u>Proof</u>. Let $G = (N, \Sigma, P, S)$ be a proper, left-recursive cfg. We use the notations given above and in algorithm 3.1., hence $N = \{A_1,A_2,\dots,A_r\}$. The cfg obtained after applying algorithm 3.1. is $G' = (N', \Sigma, P', S)$. We have to show $L(G') = L(G)$, G' right-covers G and G' is non-left-recursive. In the algorithm a sequence of grammars is obtained in the following way. The algorithm starts with cfg $G_{11} = G$, hence $i = 1$ in the algorithm. Step (2) produces cfg G_{21}, hence $i = 2$ and $j = 1$ in the algorithm. Step (4) is applied and gives cfg G_{22}. For $i = 2$ step (2) is again applied and cfg G_{31} is obtained. For $i = 3$ and $j = 1$ step (4) is applied, result G_{32}, and for $i = 3$ and $j = 2$ step (4) is once more applied and cfg G_{33} is obtained. The algorithm halts if G_{rr} has been reached.

Hence, each G_{i1}, $i > 1$, is constructed from G_{kk}, where $k = i - 1$, by applying step (2) of the algorithm. Each G_{ik}, $i > 1$ and $1 < k \leq i$, is constructed from G_{ij}, where $j = k - 1$, by applying step (4) of the algorithm.

<u>CLAIM 1</u>.

The transitions from G_{kk} to G_{i1} by step (2), where $i = k + 1$, and from G_{ij} to G_{ik} by step (4), where $k = j + 1$ and $1 < k \leq i$, are language-and cover-preserving.

<u>Proof of Claim</u> 1.

Notice that initially we start with $L(G_{11}) = L(G)$ and G_{11} right-covers $G = G_{11}$, and since the cover-relation is transitive we can obtain G' $(= G_{rr})$ right-covers G $(= G_{11})$.

First we are concerned with step (2), transition of G_{kk} to G_{i1}, $i = k + 1$. In G_{kk} we have productions $A_i \to A_i\alpha_1 | A_i\alpha_2 | \ldots A_i\alpha_m | \beta_1 | \beta_2 | \ldots | \beta_n$ which we label with $a_1, a_2, \ldots, a_m$ and $b_1, b_2, \ldots, b_n$. In G_{i1} we obtain the productions

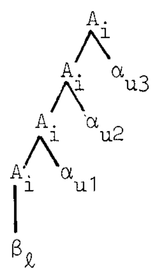
(c_1, c_2): $A_i \to C_i \mid C_iA_i'$

(d_1, d_2): $A_i' \to D_i \mid D_iA_i'$

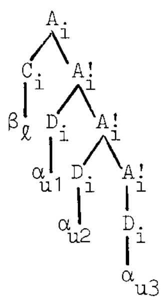
$(a_1', a_2', \ldots, a_m')$: $D_i \to \alpha_1 | \alpha_2 | \ldots | \alpha_m$

$(b_1', b_2', \ldots, b_n')$: $C_i \to \beta_1 | \beta_2 | \ldots | \beta_n$

We can verify that this transformation is language-preserving by comparing trees in G_{kk} and in G_{i1} with roots A_i and noticing that, since C_i, D_i and A_i' are new nonterminals which can only be derived from A_i, these trees can be considered independently from the rest of a parse tree.



parse tree in G_{kk}



parse tree in G_{i1}
($i = k + 1$)

Suppose we have a sentence w in $L(A_i)$, then this sentence has the form $w = s(\beta_\ell)\ s(\alpha_{u1})\ s(\alpha_{u2}) \ldots s(\alpha_{up})$, where $1 \le \ell \le n$, the α-indices are between 1 and m, and in a leftmost derivation $p + 1$ successive A_i-productions ($p \ge 0$) have been used. Let $p > 0$, then in G_{kk} a right parse for w to A_i is of the form

$rs(\beta_\ell)b_\ell\ rs(\alpha_{u1})\,a_{u1}\,rs(\alpha_{u2})\,a_{u2} \ldots rs(\alpha_{up})\,a_{up}$

and in G_{i1} we obtain for the right parse the form

$$rs(\beta_\ell)b_\ell'\ rs(\alpha_{u1})\,a_{u1}'\,rs(\alpha_{u2})\,a_{u2}' \ldots rs(\alpha_{up})a_{up}'\ d_1(d_2)^{p-1}\ c_2.$$

Hence, G_{i1} right-covers G_{kk} with cover-homomorphism h, if we define

$h(b_\ell') = b_\ell$, $1 \le \ell \le n$

$h(a_\ell') = a_\ell$, $1 \le \ell \le m$

$h(c_1) = h(c_2) = h(d_1) = h(d_2) = \varepsilon$,

where $h(c_1) = \varepsilon$ can be verified by considering the case $p = 0$. Each other production of G_{i1} is mapped on itself by h.

Now we treat the transition of a cfg G_{ij} by step (4) of the algorithm to a cfg G_{it}, where $t = j + 1$. In G_{ij} the productions $A_i \to A_j\gamma_1 | A_j\gamma_2 | \ldots | A_j\gamma_q$ are labeled with $y_1, y_2, \ldots, y_q$. We first consider case (a) of the algorithm. Hence we have in G_{ij} the productions

(c_1, c_2): $A_j \to C_j | C_jA_j'$

(d_1, d_2): $A_j' \to D_j | D_jA_j'$

$(a_1, a_2, \ldots, a_m)$: $D_j \to \alpha_1 | \alpha_2 | \ldots | \alpha_m$ †)

$(b_1, b_2, \ldots, b_n)$: $C_j \to \beta_1 | \beta_2 | \ldots | \beta_n$

†) Notice that the values of m, n and q depend on i and j of the algorithm. Since our notation will be clear we omit indices.

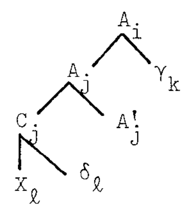
In G_{it} we obtain by step (4a), for each production (y_k): $A_i \to A_j\gamma_k$, where $1 \le k \le q$, the productions $A_i \to X_1H_j^1\gamma_k | X_2H_j^2\gamma_k | \ldots | X_nH_j^n\gamma_k$, which we label with $y_{k1}, y_{k2}, \ldots, y_{kn}$. We also obtain productions

(c_ℓ'): $H_j^\ell \to Q_j^\ell A_j'$ $\quad 1 \le \ell \le n$

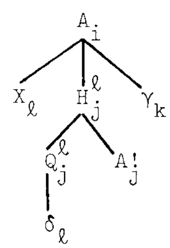
(e_ℓ): $H^\ell \to Q_j^\ell$ $\quad 1 \le \ell \le n$

(b_ℓ'): $Q_j^\ell \to \delta_\ell$ $\quad 1 \le \ell \le n$

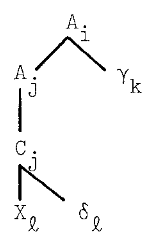
If we observe the parse trees for sentences with respect to G_{ij} and G_{it}, it is again sufficient to consider sub-trees with roots A_i.



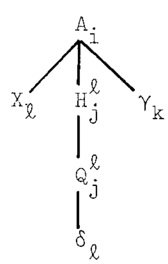
in G_{ij}



in G_{it}



in G_{ij}



in G_{it}

For every combination of i and j ($1 \le i \le r$ and $1 \le j < i$), step (4a) is done once at most. It will be clear from the possible parse trees in G_{ij} and G_{it} that the transformation in step (4a) is language preserving. To observe the cover-property we only consider the case that in the figures given above $A_j \to C_j A'_j$ is used. The case $A_j \to C_j$ can be treated similarly.

If we have a sentence w in $L(A_i)$ then it is of the form $w = s(X_\ell)\, s(\delta_\ell)\, s(A'_j)\, s(\gamma_k)$ and a right parse of w to A_i with respect to G_{ij} is of the form

$$rs(X_\ell)\ rs(\delta_\ell)\ b_\ell\ rs(A'_j)\ c_2\ rs(\gamma_k)\ y_k$$

and a right parse of w to A_i with respect to G_{it} is

$$rs(X_\ell)\ rs(\delta_\ell)\ b'_\ell\ rs(A'_j)\ c'_\ell\ rs(\gamma_k)\ y_{k\ell}$$

Now it is clear that we can define the cover- homomorphism h, such that G_{it} right-covers G_{ij}, where

$h(b'_\ell) = b_\ell, \quad 1 \le \ell \le n$

$h(e_\ell) = c_1, \quad 1 \le \ell \le n$

$h(c'_\ell) = c_2, \quad 1 \le \ell \le n$

$h(y_{k\ell}) = y_k, \quad 1 \le \ell \le n$ and $1 \le k \le q$.

Each other production of G_{it} is mapped on itself by h. The definition $h(e_\ell) = c_1$ can be verified by considering the case $A_j \to C_j$. Now we consider case (b). Hence C_j is not defined. Suppose we have the following A_j-productions in G_{ij}:

$A_j \to X_1\delta_1 | X_2\delta_2 | \ldots | X_p\delta_p$.

We label these productions with $b_1, b_2, \ldots, b_p$. For G_{it} we obtain by step (4b) for each production (y_k): $A_i \to A_j\gamma_k$, where $1 \le k \le q$, the productions (with labels $y_{k1}, y_{k2}, \ldots, y_{kp}$)

$A_i \to X_1H_j^1\gamma_k | X_2H_j^2\gamma_k | \ldots | X_pH_j^p\gamma_k$, and we obtain also the productions (b'_ℓ): $H_j^\ell \to \delta_\ell$, $1 \le \ell \le p$.

Now, in the same way as was done in case (a) one can verify that, to obtain a cover-homomorphism one has to define $h(b'_\ell) = b_\ell$, $1 \le \ell \le p$, $h(y_{k\ell}) = y_k$, $1 \le \ell \le p$ and $1 \le k \le q$,

and each other production of G_{it} has to be mapped on itself by h. Since now we can conclude that step (2) and step (4) are language- and cover-preserving we conclude G' right-covers G. Notice that with algorithm 3.1. we obtain immediately the cover-homomorphism for G' and G, since every production obtained after a transformation is immediately related to the production of grammar G, as indicated between parentheses after each production in the algorithm.

<u>CLAIM 2.</u>

G' is non-left-recursive.

<u>Proof of Claim 2.</u>

First we notice that, since δ_ℓ obtained in step (4) of the algorithm may be the empty string, G' does not have to be proper. We make the following observations.

<u>OBSERVATION 1.</u> Let $L(A_i)$ and $L'(A_i)$ denote the languages obtained from A_i in G and G' respectively. The transformations on the productions in step (2) and in step (4) are such that for each i we have $L(A_i) = L'(A_i)$. Since G is a proper grammar we have in G' $A_i \stackrel{*}{\not\Rightarrow} \varepsilon$ and $C_i \stackrel{*}{\not\Rightarrow} \varepsilon$, for each A_i and C_i.

<u>OBSERVATION 2.</u> For each D_i, introduced in step (2), we have $D_i \stackrel{*}{\not\Rightarrow} \varepsilon$. To show this we first prove the following property. Suppose we are in algorithm 3.1. at the moment we want to do step (2) for nonterminal A_i. Let $A_i \to A_k\gamma_k$ be a production in G_{ii}, where $i \le k$. Then we have the following property: if $\gamma_k \stackrel{*}{\Longrightarrow} \varepsilon$ in G_{ii}, then $A_i \stackrel{+}{\Longrightarrow} A_k$ in G.

The proof of this property is by induction on i.

Basis. Let i = 1, then $A_1 \to A_k\gamma_k$, for $k \ge 1$, is also a production in G. If $\gamma_k \stackrel{*}{\Longrightarrow} \varepsilon$, then since G is proper we have $A_1 \to A_k$, hence $A_1 \stackrel{+}{\Longrightarrow} A_k$.

Induction. Suppose this property holds for all p such that $p < i$. We prove that we may conclude that we may conclude that this property also holds for i. Consider $A_i \to A_k\gamma_k$, where $i \le k$, in G_{ii}. If $A_i \to A_k\gamma_k$ is also in G, then we have $A_i \stackrel{+}{\Longrightarrow} A_k$ in G if $\gamma_k \stackrel{*}{\Longrightarrow} \varepsilon$.

Now assume $A_i \to A_k\gamma_k$ is not in G. Hence this production is constructed in step (4) of the algorithm and therefore it is of the form

$A_i \to A_k\ H_{jn}\ \ldots\ H_{j2}H_{j1}\delta_i$, where $n < k$. *)

To obtain this production we started with a production $A_i \to A_{j1}\delta_i$ in G, where $i > j1$, and in

*) Notice that the use of indices here is somewhat different of the use in the algorithm. If they are not necessary we omit the upper-indices of the nonterminals H and Q (see step (4)). By ji is meant j with index i.

step (4) we used successively the productions **) $A_{j1} \to A_{j2}\gamma_1$, $A_{j2} \to A_{j3}\gamma_2$, ..., $A_{jn} \to A_k\gamma_n$ of $G_{j1,j1}$, $G_{j2,j2}$, ..., $G_{jn,jn}$ respectively.
According to step (4) we have $jp < jq$ if $p < q$, and thus by the induction hypothesis $A_{j\ell} \xRightarrow{+} A_{j(\ell+1)}$ for $1 \le \ell \le n$ and $A_k = A_{j(n+1)}$, if $\gamma_\ell \xRightarrow{*} \varepsilon$, $1 \le \ell \le n$. Therefore we have $A_i \xRightarrow{+} A_k$ in G if $\delta_i = \varepsilon$ and all $\gamma_i \xRightarrow{*} \varepsilon$, $1 \le i \le n$ and this completes the proof of the property.
Now let $k = i$ in this property, then if $D_i \xRightarrow{+} \varepsilon$ in $G_{i+1,1}$ and hence in G', we obtain $A_i \xRightarrow{+} A_i$ in G, which contradicts the fact that G is a proper context-free grammar.

OBSERVATION 3. For each A_i we have A_i is not left-recursive. We prove this also by induction. Consider the following two properties of the algorithm.

(4.1) After step (2) is executed for i, all A_i-productions begin with either
  (a) a terminal or a nonterminal A_k, $k > i$, or
  (b) C_i and the C_i-productions begin with a terminal or with a nonterminal A_k, $k > i$.

(4.2) After step (4) is executed for i and j, all A_i-productions begin with a terminal or with a nonterminal A_k, for $k > j$.

Recall that $N = \{A_1, A_2, \ldots, A_r\}$. In the proof m is the number of A_i-productions of which the right-hand sides begin with A_i (see step (2) of algorithm 3.1).

We define the *score* of an instance of (4.1) to be r.i. The score of an instance of (4.2) is $r.(i-1) + j$, where $1 \le j < i$. We prove (4.1) and (4.2) by induction on the score of an instance of these statements.

Basis. For $i = 1$ we only have instance (4.1). The transformation in step (2) is indeed such that, if $m = 0$, all A_i-productions begin with a terminal or with a nonterminal A_k, for $k > i$. And if $m > 0$ then the C_i-productions begin with a terminal or a nonterminal A_k, for $k > i$.

Induction.

(1) Assume (4.1) and (4.2) for scores less than s, and let i and j be such that $0 < j < i \le r$ and $r.(i-1) + j = s$. Since $r.j < s$ all A_j-productions begin with either
  (a) a terminal or A_k, for $k > j$, or
  (b) C_j and the C_j-productions begin with a terminal or A_k, for $k > j$.

  Since the transformation in step (4) is such that each new A_i-production begins with the begin-symbol of an A_j- (or C_j-) production and, if this symbol is a nonterminal A_k then $k > j$, we see that each A_i-production begins with a terminal or a nonterminal A_k, for $k > j$.

(2) Assume (4.1) and (4.2) for scores less than s, and let i be such that $i.r = s$. Since $(i-1).r + j < s$ we have that all A_i-productions begin with a terminal or a nonterminal A_k, for $k > j$, hence for $k \ge i$. If in step (2) $m = 0$ we have that all A_i-productions begin with a terminal or a nonterminal A_k, for $k > i$. If $m > 0$ we see that after the transformation all C_i-productions begin with the first symbol of the A_i-productions which begin with a terminal or with a nonterminal A_k, for $k > i$.
  This completes the proof that each A_i is not left-recursive.

OBSERVATION 4. From the two properties in observation 3 it is clear that, for each i, A_i cannot left-derive a nonterminal H_j, $0 \le j < r$. From observation 1 and 2 it follows that, for each i, neither C_i nor D_i can derive ε. Moreover D_i cannot left-derive H_j, $0 \le j < r$, since this would mean there is a nonterminal A_k, $0 \le k \le r$, which can left-derive H_j.
Nonterminal A_i' can only be introduced in a derivation by the productions $A_i \to C_iA_i'$, $A_i' \to D_iA_i'$ or $H_i \to Q_iA_i'$. Productions with left-hand side A_i' are $A_i' \to D_iA_i'$ and $A_i' \to D_i$. Since $C_i \overset{*}{\not\Rightarrow} \varepsilon$ and $D_i \overset{*}{\not\Rightarrow} \varepsilon$ the only possibility for A_i' to be left-recursive is that A_i' can left-derive H_i and $Q_i \xRightarrow{*} \varepsilon$. However, then also D_i can left-derive H_i, which is not true. Therefore, for each i, A_i' and

**) For convenience we give the productions in $G_{j1,j1}$, $G_{j2,j2}$ etc., instead of the eventually ultimate productions $A_{j1} \to C_{j1}$ and $C_{j1} \to A_{j2}\gamma_1$ etc.

also D_i are not left-recursive. Easily can be verified that, for each i, C_i is not left-recursive.

OBSERVATION 5. For each i we have H_i and Q_i are not left-recursive (we omit again the upper indices). The proof of this statement is by induction on i. First we assume that the δ's in step (4) are not equal to ε.

Basis. Let j be the smallest integer such that H_j is defined. Let $A_j \to X\delta$ (if m = 0 in step (2)) or $A_j \to C_j$ and $C_j \to X\delta$ (if m > 0 in step (2)), then we obtain $H_j \to \delta$ or $H_j \to Q_j | Q_j A'_j$ and $Q_j \to \delta$ respectively. Since there are no other nonterminals H_p, $1 \le p \le r$, defined before, δ can only begin with a terminal or a nonterminal A_k, $1 \le k \le r$. A nonterminal A_k cannot left-derive a nonterminal H_p, $1 \le p \le r$.

Induction. We prove that H_i cannot left-derive a nonterminal H_p, if $p \ge i$. Therefore we assume that the nonterminals H_t, for $t < i$, cannot left-derive a nonterminal H_p if $p \ge t$.

Suppose H_i is introduced for an A_ℓ-production, that is, in step (4) we transformed a production $A_\ell \to A_q\gamma$ (where $q < \ell$) of G, and after all steps (4) for this production have been executed the result is $A_\ell \to XH_i\gamma'$ where X is a terminal or a nonterminal A_k for $k \ge \ell$. The last production which was applied in step (4) is then of the form $A_i \to X\delta$ (or $A_i \to C_i$ and $C_i \to X\delta$). Moreover we obtain the production $H_i \to \delta$ or $H_i \to Q_i A'_i | Q_i$ and $Q_i \to \delta$. If δ begins with a terminal or a nonterminal A_k, $1 \le k \le r$, then, since A_k cannot left-derive a nonterminal H_p, $1 \le p \le r$, H_i is not left-recursive. The other possibility is that δ begins with a nonterminal H_p, where $p < i$. Then by the induction hypothesis H_p cannot left-derive H_i. This completes the proof of the induction step.

Now assume $\delta = \varepsilon$. Then we can have $H_i \stackrel{+}{\Longrightarrow} Q_i A'_i \stackrel{+}{\Longrightarrow} A'_i$. However A'_i cannot left-derive H_i. It is easily possible to give a proof of this statement analogous to the proof given above, where instead of the δ's of step (4) we have to consider the α's of step (2).

Since the nonterminals H_i are not left-recursive we immediately obtain that the nonterminals Q_i are not left-recursive. This completes the proof of observation 5.

Since we must conclude that all the nonterminals are non-left-recursive we have finished the proof of claim 2 and therefore of theorem 3.1.□

Before closing this section we give an example of an application of algorithm 3.1. We use a grammar which was also used in [1, p.157]. The cover-homomorphism of G' to G is obtained between parentheses after each production. Hence, we immediately relate every production obtained in the transformation, to a production of the original grammar G.

EXAMPLE 3.1

Consider the cfg G with productions

1.	$A_1 \to A_2A_3$	(1)	5.	$A_3 \to A_1A_2$	(5)
2.	$A_1 \to a$	(2)	6.	$A_3 \to A_3A_3$	(6)
3.	$A_2 \to A_3A_1$	(3)	7.	$A_3 \to a$	(7)
4.	$A_2 \to A_1b$	(4)			

We follow the steps of the algorithm.

(1) i = 1

(2) $A_1 \to A_2A_3 | a$ (1,2) remains unaltered

(3) i = 2, j = 1

(4) Replace $A_2 \to A_1b$ (4), where $A_1 \to A_2A_3 | a$ (1,2)

by $A_2 \to A_2H_1^1b$ (4)

$A_2 \to aH_1^2b$ (4)

$H_1^1 \to A_3$ (1)

$H_1^2 \to \varepsilon$ (2)

(2) Replace $A_2 \to A_2H_1^1b \mid aH_1^2b \mid A_3A_1$ (4,4,3)

by $A_2 \to C_2 \mid C_2A'_2$ $(\varepsilon,\varepsilon)$

$A'_2 \to D_2 \mid D_2A'_2$ $(\varepsilon,\varepsilon)$

$D_2 \to H_1^1b$ (4)

$C_2 \to aH_1^2b \mid A_3A_1$ (4,3)

(3) i = 3, j = 1

(4) Replace $A_3 \to A_1A_2$ (5), where $A_1 \to A_2A_3 | a$ (1,2)

by $A_3 \to A_2H_1^1A_2$ (5)

$A_3 \to aH_1^2A_2$ (5)

(5) j = 2

(4) Replace $A_3 \to A_2H_1^1A_2$ (5), where $A_2 \to C_2 | C_2A'_2$ $(\varepsilon,\varepsilon)$

and $C_2 \to aH_1^2b | A_3A_1$ (4,3)

by $A_3 \to A_3H_2^1H_1^1A_2$ (5)

$$\begin{array}{ll}
A_3 \rightarrow aH_1^2H_1^1A_2 & (5) \\
H_2^1 \rightarrow Q_2^1A_2' & (\varepsilon) \\
H_2^1 \rightarrow Q_2^1 & (\varepsilon) \\
Q_2^1 \rightarrow A_1 & (3) \\
H_2^2 \rightarrow Q_2^2A_2' & (\varepsilon) \\
H_2^2 \rightarrow Q_2^2 & (\varepsilon) \\
Q_2^2 \rightarrow H_1^2b & (4)
\end{array}$$

(2) Replace

$$A_3 \rightarrow A_3H_2^1H_1^1A_2 \mid A_3A_3 \mid aH_2^2H_1^1A_2 \mid aH_1^2A_2 \mid a \quad (5,6,5,5,7)$$

by

$$\begin{array}{ll}
A_3 \rightarrow C_3 \mid C_3A_3' & (\varepsilon,\varepsilon) \\
A_3' \rightarrow D_3 \mid D_3A_3' & (\varepsilon,\varepsilon) \\
D_3 \rightarrow H_2^1H_1^1A_2 \mid A_3 & (5,6) \\
C_3 \rightarrow aH_2^2H_1^1A_2 \mid aH_1^2A_2 \mid a & (5,5,7)
\end{array}$$

The resulting grammar G' (see below) has 26 productions while the original grammar had 7 productions. The usual method yields 22 productions. The usual method was given in section 2. The cfg G' has the following productions:

$$\begin{array}{ll}
A_1 \rightarrow A_2A_3 \mid a & (1,2) \\
A_2 \rightarrow C_2 \mid C_2A_2' & (\varepsilon,\varepsilon) \\
A_2' \rightarrow D_2 \mid D_2A_2' & (\varepsilon,\varepsilon) \\
D_2 \rightarrow H_1^1b & (4) \\
C_2 \rightarrow aH_1^2b \mid A_3A_1 & (4,3) \\
H_1^1 \rightarrow A_3 & (1) \\
H_1^2 \rightarrow \varepsilon & (2) \\
A_3 \rightarrow C_3 \mid C_3A_3' & (\varepsilon,\varepsilon) \\
A_3' \rightarrow D_3 \mid D_3A_3' & (\varepsilon,\varepsilon) \\
D_3 \rightarrow H_2^1H_1^1A_2 \mid A_3 & (5,6) \\
C_3 \rightarrow aH_2^2H_1^1A_2 \mid aH_1^2A_2 \mid a & (5,5,7) \\
H_2^1 \rightarrow Q_2^1A_2' \mid Q_2^1 & (\varepsilon,\varepsilon) \\
H_2^2 \rightarrow Q_2^2A_2' \mid Q_2^2 & (\varepsilon,\varepsilon) \\
Q_2^1 \rightarrow A_1 & (3) \\
Q_2^2 \rightarrow H_1^2b & (4)
\end{array}$$

Notice that G' is not proper since $H_1^2 \rightarrow \varepsilon$.

4. LEFT-TO-RIGHT COVER

In the preceding section we saw that each left-recursive grammar G can be right-covered by a non-left-recursive grammar G'. If we look at the parsing problem then we want to eliminate left-recursion since a certain top-down parsing method will not work for a left-recursive grammar. We want to make the grammar fitting for this top-down parsing method, and this means in general for a parsing method which produces left parses. However our algorithm is only concerned with right parses. Fortunately we can give the following theorem. This theorem can also be found in [3] in a slightly different form.

THEOREM 4.1.
Let cfg G' right-cover G, then there is a cfg G", such that G" left-to-right covers G.

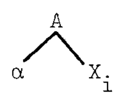
Proof. Let $G' = (N', \Sigma, P', S')$ right-cover $G = (N, \Sigma, P, S)$ by cover-homomorphism h. We construct a new grammar $G'' = (N'', \Sigma, P'', S'')$, $S'' = S'$ and where $N'' = N' \cup \{R_i \mid 1 \leq i \leq |P'|\}$ and each R_i is not already in N'.

$P'' = P_1 \cup P_2$, where

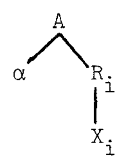
$P_1 = \{(i'){:}\ A \rightarrow \alpha R_i \mid (i){:}\ A \rightarrow \alpha X_i \text{ is in } P'\}$ and

$P_2 = \{(i''){:}\ R_i \rightarrow X_i \mid (i'){:}\ A \rightarrow \alpha R_i \text{ is in } P_1\}$.

If T' is a parse tree of G' for a sentence w then there is a corresponding parse tree T" of G" for w which is obtained from T' by replacing each occurrence of a subtree T_o' in T' of the form



by a subtree T_o'' of T" of the form



These occurrences of subtrees in T' and T" of these forms are said to be corresponding. In T_o'' the productions (i'): $A \rightarrow \alpha R_i$ and (i''): $R_i \rightarrow X_i$ are said to be connected. Notice, that if π' is a parse of w with respect to T' and π'' is a parse of w with respect to T" then for each occurrence of i in π' there is only one corresponding pair i' and i" in π''. Similarly, for each occurrence of i' in π'' there is only one connected occurrence of i".

Now the proof is rather simple. Let T' be a parse tree of G' for w and T" its corresponding parse tree of G". A left parse π" for w with respect to T" in which productions i' and j' occur can be written in one of the following forms:

a". π" ≡ ...i'...j'...j"...i"..., or

b". π" ≡ ...i'...i"...j'...j"...,

or symmetric cases (first j'), where i' and i" are connected and j' and j" are connected. For these cases the right parses with respect to T' can be written as:

a'. π' ≡ ...j...i... (for case a".), and

b'. π' ...i...j... (for case b".),

where i corresponds to the connected pair i' and i" and j corresponds to the connected pair j' and j". For i' ≠ j' a form

c". π" ≡ ...i'...j'...i"...j"...

cannot occur in a left parse. For i' = j' there are no problems as can be seen in what follows. Since the order in which i" and j" appear in π" is the same as the order in which i and j appear in π' we can define a homomorphism h' such that, for each i' and i", h'(i") = i and h'(i') = ε and then G" left-to-right covers G' with cover-homomorphism h'. The composition of h' and h gives the cover-homomorphism f of G" left-to-right covers G, that is, f(i") = h(h'(i")) and f(i') = ε for each pair i" and i' in P". □

Since in this theorem G" is not left-recursive if G' is not left-recursive a top-down parsing method can be used for G" and the left parses with respect to G" can be mapped on the right parses with respect to G.

5. CONCLUSIONS.

We showed that some remarks concerning left recursion in the literature are not true. An algorithm was given to transform a left-recursive grammar G to an non-left-recursive grammar G' such that G' right-covers G. We showed that the use of right parses in this algorithm is not restrictive in a practical situation in which we want to eliminate left-recursion to have the possibility to apply a top-down parsing method which yields left parses.

There are some problems we did not consider. Can the elimination of ε-productions be done in such a way that the result is a covering grammar? According to some remarks in [1], that we gave in section 2, we can conclude that if a cfg is ambiguous then elimination of ε-productions can not lead in general to a covering grammar, and if a cfg is unambiguous then there is a covering grammar. However, the following grammar with productions $S \to LS0|LS1|0|1$ and $L \to \varepsilon$ is not ambiguous and we conjecture that this grammar is not right-covered by an ε-free grammar. Furthermore we can ask to prove the conjecture that grammar G_1 of section 2 cannot be right-covered by a cfg in GNF even if we do not restrict ourselves to a fine cover-homomorphism.

ACKNOWLEDGEMENT

I am grateful to prof. L.A.M. Verbeek for some helpful comments. The research reported in this paper has been carried out at the Twente University of Technology.

I thank ms. Marja Verburg for her beautiful and careful typing of the manuscript.

REFERENCES.

1. Aho A.V. and Ullman J.D., *"The theory of parsing, translation and compiling", Vol. I and II, Prentice Hall, Englewood Cliffs, 1972 and 1973.*

2. Gray J. and Harrison M.A., *"On the covering and reduction problems for context-free grammars", J. Assoc. Comput. Mach. 19, (1972), No.4, 675 - 698.*

3. Nijholt A., *"On the covering of parsable grammars", to appear in J. Comp. Syst. Sci.*