Reprinted from JOURNAL OF COMPUTER AND SYSTEM SCIENCES
All Rights Reserved by Academic Press, New York and London

Vol. 15, No. 1, August 1977
Printed in Belgium

On the Covering of Parsable Grammars

ANTON NIJHOLT*

Department of Applied Mathematics, Twente University of Technology, Enschede, The Netherlands

Received May 4, 1976; revised November 30, 1976

The notion of a parsable grammar is introduced. A definition of cover is provided which is a generalization of a well-known definition of cover. With this new definition of cover we prove that every parsable grammar is covered by an $LR(1)$ grammar or, if the language is prefix-free, by a strict deterministic grammar. A consequence of this result is that every $LR(k)$ grammar is covered by an $LR(1)$ or a strict deterministic grammar.

1. INTRODUCTION

Numerous parsing methods have been introduced in the literature. Each of these methods also defines a certain class of grammars for which the method works.

It is natural to ask for the possibility to transform a grammar belonging to one class into a grammar belonging to another class. If we consider the parsing problem there are two conditions which have to be fulfilled. The grammar obtained by the transformation should generate the same language, and it has for each sentence a parse which is equal to or can easily be converted to the parse of the same sentence in the original grammar.

In this way it is possible to transform grammars which are difficult to parse into grammars which can be parsed more easily. This is illustrated in Fig. 1. The conversion of one parse to another parse is formalized (e.g. in [1, 3]) to a definition of covering of grammars. Mickunas and Schneider [10] gave a direct transformation of an $LR(k)$ grammar to an $LR(1)$ grammar (or under additional hypotheses to a $LR(0)$ grammar) such that the main

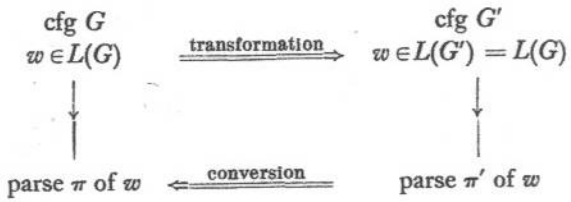

FIGURE 1

* Author's present address: Department of Mathematics, Free University, P.O. Box 7161, Amsterdam, The Netherlands.

ISSN 0022-0000

part of the original grammar is covered by the new grammar. A very complicated proof of this result appeared in [9]. Moreover, Mickunas [8] presented another algorithm for this transformation. In this paper we show that the proof of the property that every $LR(k)$ grammar is covered by an $LR(1)$ or $LR(0)$ grammar is rather trivial by using the standard conversion of a deterministic pushdown tranducer to a context-free grammar. This problem was presented as an open problem in [1, p. 709] and in [5].

The remainder of this section is devoted to some basic concepts concerning formal grammars and automata.

DEFINITION 1.1. A *context-free grammar* (*cfg* for short) is a four-tuple $G = (N, \Sigma, P, S)$, where N and Σ are two alphabets, $N \cap \Sigma = \phi$ (letters in Σ and N are called *terminals* and *nonterminals*, respectively), $V = N \cup \Sigma$, $S \in N$ and the set of *productions* P is a finite subset of $N \times V^*$. The productions in P are numbered for identification. If (A, γ) is in P then we write $A \to \gamma$ or we use the notation $i.\ A \to \gamma$ if $A \to \gamma$ is the ith production in P.

$\xi \Rightarrow \psi$ is defined for strings $\xi, \psi \in V^*$ if there exist strings $\alpha, \beta, \gamma \in V^*$ and $A \in N$ so that $\xi = \alpha A \beta$, $\psi = \alpha\gamma\beta$, and $A \to \gamma \in P$.

If $\beta \in \Sigma^*$ we may write $\xi \Rightarrow_r \psi$ and if $\alpha \in \Sigma^*$ we may write $\xi \Rightarrow_l \psi$. In the usual way we denote the transitive and the reflexive-transitive closure of $\Rightarrow$ by $\Rightarrow^+$, and $\overset{*}{\Rightarrow}$, respectively, and the same can be done for $\Rightarrow_l$ and $\Rightarrow_r$. The *language* generated by G is $L(G) = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$. A sequence $\xi_0 \Rightarrow \xi_1 \Rightarrow \cdots \Rightarrow \xi_n$, where $\xi_0 = S$ and $\xi_n = w \in \Sigma^*$, is called a *derivation* of w. Each element of this sequence is called a *sentential form*. If $\Rightarrow$ is replaced by $\Rightarrow_l$ or $\Rightarrow_r$ this derivation is said to be a *leftmost derivation* (*left parse*) or a *rightmost derivation* (*right parse*), respectively.

Sometimes we use the notation $S \Rightarrow^{\pi} w$, where π denotes a certain concatenation of numbers of productions used in the derivation of w. If $\alpha \in V^*$ then $|\alpha|$ denotes the *length* of α. The first k symbols of α are denoted by $k{:}\ \alpha$. If $|\alpha| < k$ then $k{:}\ \alpha = \alpha$. The *empty string* is denoted by ϵ. A language L is said to be *prefix-free* iff $u \in L$ and $uv \in L$ implies $v = \epsilon$.

All cfg's in this paper are *reduced*, i.e., each element of V can appear in a sentential form and each nonterminal can generate a string of terminals. A cfg G is said to be unambiguous if each sentence in $L(G)$ has only one left parse.

DEFINITION 1.2 [6]. A cfg $G = (N, \Sigma, P, S)$ is said to be *strict deterministic* if there exists a partition ρ of V such that

(1) $\Sigma \in \rho$,

(2) for any $A, A' \in N$ and $\alpha, \beta, \beta' \in V^*$, if $A \to \alpha\beta$, $A' \to \alpha\beta'$ and $A \equiv A' \pmod{\rho}$ then either

(i) both $\beta, \beta' \neq \epsilon$ and $1 : \beta \equiv 1 : \beta' \pmod{\rho}$ or

(ii) $\beta = \beta' = \epsilon$ and $A = A'$.

DEFINITION 1.3. A (reduced) cfg $G = (N, \Sigma, P, S)$ such that there is no derivation

$S \Rightarrow^+ S$ possible, is said to be an $LR(k)$ *grammar* if, for each w, w', $x \in \Sigma^*$; γ, α, α', β, $\beta' \in V^*$; A, $A' \in N$, if

(i) $S \overset{*}{\Rightarrow}_r \alpha A w \Rightarrow_r \alpha\beta w = \gamma w$ and

(ii) $S \overset{*}{\Rightarrow}_r \alpha' A' x \Rightarrow_r \alpha'\beta' x = \gamma w'$ and

(iii) $k : w = k : w'$,

then $A \to \beta = A' \to \beta'$ and $| \alpha\beta | = | \alpha'\beta' |$.

For an extensive treatment of $LR(k)$ grammars and their languages the reader is referred to [2].

Remark (see [6]). The class of strict deterministic grammars is a proper subclass of the $LR(0)$ grammars and they generate exactly the prefix-free deterministic languages.

DEFINITION 1.4. A *deterministic pushdown transducer* (dpdt for short) is an eight-tuple $P = (Q, \Sigma, \Gamma, \Delta, \delta, q_0, Z_0, F)$, where Q is a finite set of *states*, Σ, Γ, and Δ are alphabets and δ is a mapping from $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ to $Q \times \Gamma^* \times \Delta^*$ such that if $\delta(q, a, Z)$ is defined, then $\delta(q, \epsilon, Z)$ is undefined and if $\delta(q, \epsilon, Z)$ is defined, then $\delta(q, a, Z)$ is undefined for all $a \in \Sigma$. Further, $q_0 \in Q$ is the *initial state*, $Z_0 \in \Gamma$ is the *start symbol*, and $F \subseteq Q$ is the set of *accepting states*. A *configuration* of P is a four-tuple (q, w, α, y) in $Q \times \Sigma^* \times \Gamma^* \times \Delta^*$. If $\delta(q, a, Z) = (r, \alpha, z)$ we write $(q, ax, Z\gamma, y) \vdash (r, x, \alpha\gamma, yz)$. In the usual way the move $\vdash$ is extended to $\vdash^+$ and $\overset{*}{\vdash}$. The *translation* defined by P is the set $\tau(P) = \{(x, y) \mid (q_0, x, Z_0, \epsilon) \overset{*}{\vdash} (q, \epsilon, \alpha, y)$ for some $q \in F$ and $\alpha \in \Gamma^*\}$. The language accepted by P is the set $L(P) = \{x \mid (x, y) \in \tau(P)\}$. $L(P)$ is said to be a deterministic language.

2. COVERS

In this section we try to formalize the idea of conversion of one parse to another parse, as shown in Fig. 1. The following definition can be found in [1, p. 276] and also in a slightly different form in [3].

DEFINITION 2.1. A cfg $G' = (N', \Sigma, P', S')$ is said to *right-cover* a cfg $G = (N, \Sigma, P, S)$ if

1. $L(G') = L(G)$,
2. there is a homomorphism h such that
   (i) if $S' \Rightarrow_r^{\pi'} w$, then $S \Rightarrow_r^{h(\pi')} w$, and
   (ii) for all π, such that $S \Rightarrow_r^{\pi} w$, there exists π' such that $S' \Rightarrow_r^{\pi'} w$ and $h(\pi') = \pi$.

Note that in π, π', and $h(\pi')$ the concatenation of productions is such that they appear in the same order as they are applied in the rightmost derivation of w.

A similar definition can be given for leftmost derivations. We make a few observations. We note that it is possible to introduce "cover"-definitions for other types of derivations,

and also for strings of productions used in a generation of $w \in L(G)$ that do not necessarily correspond to a derivation in the usual sense. For example, the result of *left-corner parsing* [12] is a string of productions such that in general the productions cannot be applied in a derivation in the same order as they appear in that string.

EXAMPLE. Let G be a cfg with productions 1. $E \to E + T$, 2. $E \to T$, 3. $T \to T * F$, 4. $T \to F$, 5. $F \to (E)$, and 6. $F \to a$, then the string 64362156424 is the left-corner parse for sentence $a * a + (a)$. The second observation is that in this definition right parses are mapped on right parses, or in general, a parse of type x with respect to G' is mapped on a parse of type x with respect to G. It is, however, also possible to map a parse of type x with respect to G' on a parse of type y with respect to G. An example of this can be found in [4], where a transformation of grammars is given such that left parses are mapped on reversed right parses. These observations motivate our following definition. The notation $G'[x/y]G$ mean that G' covers G such that x-parses with respect to G' are mapped on y-parses with respect to G. The sentence "*$w \in L(G)$ and π is an x-parse of w with respect to G*" is abrreviated to $S \Rightarrow_x^\pi w$.

DEFINITION 2.2. Let $G' = (N', \Sigma, P', S')$ and $G = (N, \Sigma, P, S)$ be cfg's. $G'[x/y]G$ if

1. $L(G') = L(G)$,
2. there is a homomorphism h such that
   (i) if $S' \Rightarrow_x^{\pi'} w$, then $S \Rightarrow_y^{h(\pi')} w$, and
   (ii) for all π such that $S \Rightarrow_y^\pi w$, there exists π' such that $S' \Rightarrow_x^{\pi'} w$ and $h(\pi') = \pi$.

3. PARSABLE GRAMMARS

Although in the original papers not always formally presented that way, most of the parsing methods for deterministic languages can be implemented by a deterministic pushdown transducer. See for example [1], where it is shown that a k-predictive parsing algorithm for $LL(k)$ grammars and a shift-reduce parsing algorithm for $LR(k)$ grammars can be implemented by a dpdt (with an endmarker on the input). In [5, 7] the same is done for the strict deterministic grammars.

This motivates us to consider grammars for which a parsing method exists that can be implemented by a dpdt. The most general parsing method we can then consider is a dpdt which acts as a parser, that is, given a cfg G, if $w \in L(G)$ then the dpdt with input w gives a parse with respect to G as output and if $w \notin L(G)$ then the dpdt halts and gives an error-message. Moreover, we demand that in a final state of the dpdt no moves are possible. Such a dpdt will be called a *valid dpdt* for G.

DEFINITION 3.1. A cfg G is said to be a *parsable grammar* if there exists a valid dpdt for G.

For each type of parse a subclass of the class of parsable grammars can be defined. For instance the class of *left-corner parsable* grammars is the class of those grammars for which there is a valid dpdt which gives a left-corner parse as output. In [1] the *left-parsable* and *reversed right-parsable* grammars were already defined with the aid of a simple syntax directed translation scheme. For more general types of parses this way of defining is more restrictive than with the aid of a dpdt.

For examples showing this and also for examples of grammars belonging to other classes the reader is referred to [11].

In the inclusion diagram shown in Fig. 2, U stands for unambiguous, P for parsable, LP for left parsable, and $\bar{\text{R}}$P for reversed right-parsable grammars. An example of a grammar which is parsable but which is not left or reversed right parsable is the cfg with productions 1. $S \to ABc$, 2. $S \to DBd$, 3. $A \to a$, 4. $D \to a$, 5. $B \to bBa$, and 6. $B \to b$. $L(G) = L_1 \cup L_2$, where $L_1 = \{ab^{n+1}a^nc \mid n \geqslant 0\}$, and $L_2 = \{ab^{n+1}a^nd \mid n \geqslant 0\}$. It can easily be seen that a dpdt P can be constructed such that each sentence of L_1 has a parse in $\{65^n31 \mid n \geqslant 0\}$ and each sentence in L_2 has a parse in $\{65^n42 \mid n \geqslant 0\}$ which are both sets of reversed left parses. A cfg which is unambiguous but which is not a parsable grammar is the cfg with productions 1. $S \to AEc$, 2. $S \to DB$, 3. $A \to a$, 4. $D \to a$, 5. $E \to bEc$, 6. $B \to bBc$, 7. $E \to bc$, and 8. $B \to bc$. Examples of grammars which are $LL(k)$ or $LR(k)$ but not left parsable, respectively reversed right parsable can be found in [1].

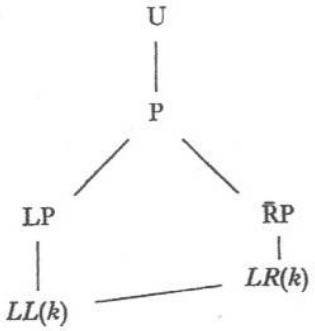

FIGURE 2

4. ON THE COVERING OF PARSABLE GRAMMARS

In this section we show that every parsable grammar is covered by a strict deterministic grammar. If cfg G is a parsable grammar then there exists a valid dpdt for G. Without loss of generality we may assume that $L(G)$ is prefix-free since, instead of parsing $w \in L(G)$, we can parse $w\perp$, where the endmarker $\perp$ is a symbol not already in the alphabet of G. The following lemma is now elementary.

LEMMA 4.1. *Let* $P = (Q, \Sigma, \Gamma, \Delta, \delta, q_0, Z_0, F)$ *be a valid* dpdt *for a* cfg G. *Then we can construct a valid* dpdt P' *for* cfg G such that $\tau(P) = \tau(P')$ *and* P' *accepts with empty pushdown list in only one final state.*

Proof. Note that in general there are some interpretations possible for a final state of the dpdt P. For example, it is possible that the sequence of configurations

$$(q_0, w, Z_0, \epsilon) \vdash^* (q, \epsilon, \alpha, y_1) \vdash^* (p, \epsilon, \alpha', y_1 y_2),$$

exists where both q and p are final states and hence according to Definition 1.4 both (w, y_1) and $(w, y_1 y_2)$ are elements of $\tau(P)$. This is not quite what we want and therefore in conformity with what we consider to be a final state of a parsing algorithm, we have demanded in the preceding section that in a final state of a valid dpdt for a cfg G no moves are possible. Notice that $L(G)$ is assumed to be prefix-free. Now construct a valid dpdt $P' = (Q \cup Q_1 \cup \{q_e\}, \Sigma, \Gamma \cup \{Z_{00}\}, \Delta, \delta', q_0, Z_{00}, \{q_e\})$, where $Q_1 = \{q' \mid q \in F\}$ and where q_e and the states $q' \in Q_1$ are newly introduced states. Z_{00} is a symbol not already in Γ and δ' is equal to δ except for the following cases.

(a) Define $\delta'(q_0, \epsilon, Z_{00}) = (q_0, Z_0 Z_{00}, \epsilon)$.

(b) For all $q \in F$, for all corresponding $q' \in Q$, and for all $X \in \Gamma$ define $\delta'(q, \epsilon, X) = (q', \epsilon, \epsilon)$, $\delta'(q', \epsilon, X) = (q', \epsilon, \epsilon)$ and $\delta'(q', \epsilon, Z_{00}) = (q_e, \epsilon, \epsilon)$.

It can easily be verified that P' satisfies the desired condition. ∎

For convenience we repeat the construction of a strict deterministic grammar from a dpdt as given by Harrison and Havel [6]. We let the productions follow by the output of the dpdt which corresponds to these productions.

CONSTRUCTION 4.1. Let $P = (Q, \Sigma, \Gamma, \Delta, \delta, q_0, Z_0, \{q_e\})$ be a valid dpdt for a cfg G which accepts with empty pushdown list and which has only one final state. Then define $G' = (N', \Sigma, P', S')$ where

(1) $N' = \{[pAa] \mid p, q \in Q, A \in \Gamma\}$, $S' = [q_0 Z_0 q_e]$;

(2) P' is defined as follows. Let $\delta(p, a, A) = (r, X_1 \cdots X_k, y)$ with $a \in \Sigma \cup \{\epsilon\}$. Then if $k > 0$ P' contains the productions $[pAq_k] \to a[rX_1q_1] \cdots [q_{k-1}X_kq_k]/(y)$ for all sequences $q_1, q_2, \ldots, q_k$ of states in Q. If $k = 0$ then the production obtained is $[pAr] \to a/(y)$.

In the sequel we assume that in a grammar obtained from this construction all useless symbols and productions are removed. With this construction we obtain a strict deterministic grammar G' such that $L(G') = L(P)$. As can be seen in step (2) G' simulates with leftmost derivations the moves that can be done by the dpdt. Therefore we have the following theorem.

THEOREM 4.1. *Let G be a parsable grammar with a valid* dpdt *which produces x-parses. Then there exists a strict deterministic grammar G' such that $G'[l/x]G$.*

Proof. Let cfg $G = (N, \Sigma, P, S)$ be a parsable grammar for a valid dpdt and parses of type x. This valid dpdt can be transformed to a valid dpdt $P = (Q, \Sigma, \Gamma, \Delta, \delta, q_0, Z_0, \{q_e\})$ which accepts with empty pushdown list in one final state. In this eight-tuple the alphabet of output symbols Δ consists of the numbers of the productions of G. From P we can construct a strict deterministic grammar $G' = (N', \Sigma, P', S')$.

Now $G'[l/x]G$, where the cover-homomorphism h is defined as follows. Every production of G' which is obtained in Construction 4.1 from $\delta(p, a, A) = (r, X_1 \cdots X_k, y)$, where $a \in \Sigma \cup \{\epsilon\}$ and $k \geqslant 0$, is mapped on y (note that $y \in \Delta^*$).

Now it is straightforward to show $S' \Rightarrow_l^{\pi'} w$ iff $(q_0, w, Z_0, \epsilon) \overset{*}{\vdash} (q_e, \epsilon, \epsilon, h(\pi'))$, and therefore we have

(a) if $S' \Rightarrow_l^{\pi'} w$ then $(q_0, w, Z_0, \epsilon) \overset{*}{\vdash} (q_e, \epsilon, \epsilon, h(\pi'))$ and since P is a valid dpdt for G we have $S \Rightarrow_x^{h(\pi')} w$;

(b) if $S \Rightarrow_x^{\pi} w$ in G then, since P is a valid dpdt for G we have $(q_0, w, Z_0, \epsilon) \overset{*}{\vdash} (q_e, \epsilon, \epsilon, \pi)$ and hence $S' \Rightarrow_l^{\pi} w$, where $h(\pi') = \pi$.

According to Definition 2.2 we conclude $G'[l/x]G$. ∎

However this result is not quite satisfactory since if a sentence is parsed with a strict deterministic parsing method the result is a reversed right parse while in this theorem we used a left parse. The next theorem gives a better result; $\bar{r}$ is used to denote reversed right parses.

THEOREM 4.2. *Let G be a parsable grammar with a valid* dpdt *which produces x-parses. Then there exists a strict deterministic grammar G' such that $G'[\bar{r}/x]G$.*

Proof. Our starting point is again a valid dpdt for G which is transformed to a valid dpdt $P = (Q, \Sigma, \Gamma, \Delta, \delta, q_0, Z_0, \{q_e\})$ for G which accepts with empty pushdown list and in only one final state. Let t be the total number of three-tuples for which δ is defined, then let $L = \{1, 2, \ldots, t\}$. The elements of L are used as labels. From P we construct a new valid dpdt $P' = (Q, \Sigma, \Gamma \cup \{K_i \mid i \in L\}, \Delta, \delta', q_0, Z_0, \{q_e\})$ where $\{K_i \mid i \in L\} \cap \Gamma = \varnothing$ and δ' is defined as follows. Suppose we have in P for $k > 0$ and $a \in \Sigma \cup \{\epsilon\}$

i. $\delta(p, a, A) = (r, X_1 \cdots X_k, y)$ (type-0 step)

then define in P'

i'. $\delta'(p, a, A) = (r, K_i X_1 \cdots X_k, \epsilon)$ (type-1 step)

and

i''. $\delta'(r, \epsilon, K_i) = (r, \epsilon, y)$. (type-2 step)

Note that we placed the output y in i''. This will be motivated later. If we have in P for $a \in \Sigma \cup \{\epsilon\}$

j. $\delta(p, a, A) = (r, \epsilon, z)$ (type-3 step)

then define in P'

j'. $\delta'(p, a, A) = (r, \epsilon, z)$. (type-4 step)

Notice that the final state of P is reached with an application of a type-3 step, with $r = q_e$, which remains unaltered. Therefore P' has the same properties as P (i.e., P' accepts with empty pushdown list and in only one final state) and $\tau(P') = \tau(P)$. Conversion of P' to a cfg yields again a strict deterministic grammar. Now consider two gram-

mars, G_1 obtained from P and G_2 obtained from P'. If for G_1 l is a production obtained from a type-0 step, that is

l. $[pAq] \rightarrow a[rX_1q_1] \cdots [q_{k-1}X_kq]/(y)$

(l is a type-0 production), then for G_2 there are productions l′ and l″ obtained from a type-1 step and a type-2 step, respectively, where

l′. $[pAq] \rightarrow a[rK_ir][rX_1q_1] \cdots [q_{k-1}X_kq]/(\epsilon)$

and

l″. $[rK_ir] \rightarrow \epsilon/(y)$

(where l′ is a type-1 production and l″ is a type-2 production). If for G_1 n is a production obtained from a type-3 step, that is

n. $[pAr] \rightarrow \epsilon/(z)$

(n is a type-3 production), then for G_2 there is a production n′ obtained from a type-4 step, where

n′. $[pAr] \rightarrow \epsilon/(z)$

(n′ is a type-4 production).

Notice that each occurrence of a subtree only consisting of a type-0 production l in a parse tree with respect to G_1 has a *corresponding* occurrence of a subtree only consisting of l′ and l″ in the parse tree with respect to G_2.

The occurrences of l′ and l″ in such a corresponding subtree are said to be *connected*. Instead of saying that such subtrees correspond we will also say that in such a case l corresponds with l′ and l″ or simply with l″. Now consider a reversed right parse π, with respect to G_2 of a sentence w, in which we have occurrences of productions l′, l″, m′, and m'' (type-1 and type-2 productions). That is π can be written as one of the two forms

a. $\pi = \cdots \text{l}'' \cdots m'' \cdots \text{m}' \cdots \text{l}' \cdots,$

b. $\pi = \cdots \text{l}'' \cdots \text{l}' \cdots \text{m}'' \cdots \text{m}' \cdots,$

where l″ and l′ are connected and m'' and m′ are connected. For $\text{l}' \neq \text{m}'$ a form $\cdots \text{l}'' \cdots \text{m}'' \cdots \text{l}' \cdots \text{m}' \cdots$ cannot exist for connected occurrences of l′ and l″. The situation $\text{l}' = \text{m}'$ will not lead to problems as can be seen in what follows.

In a left parse of w with respect to G_1 both forms a and b change to

c. $\pi_1 = \cdots \text{l} \cdots \text{m} \cdots$

where l corresponds to the occurrences of l′ and l″ displayed above in a and b, and m corresponds to the occurrences of m′ and m″ displayed above in a and b. Notice that l and m in c appear in the same order as l″ and m″ in both a and b, which will make it possible to us to define a cover-homomorphism.

In the same way as we did above for type-0, type-1, and type-2 productions we can consider type-3 and type-4 productions or combinations, and the following claim can be easily verified.

Claim. The order of type-2 and type-4 productions in a reversed right parse of a sentence w with respect to G_2 is the same as the order of the corresponding type-0 and type-3 productions in the left parse with respect to G_1.

Our intentions will be clear. We want to map reversed right parses with respect to G_2 on the x-parses with respect to G and we make use of cfg G_1 since we know that $G_1[l/x]G$. There remains a small problem. In general we do not have $G_2[\bar{r}/l]\, G_1$ since a type-2 production can correspond to more than one type-0 production. However, for a given type-2 production each of the corresponding type-0 productions is obtained from the *same* type-0 step, which means that for the cover-homomorphism of $G_1[l/x]G$ each of these productions is mapped on the output, say y, given in this step. Therefore we can immediately define a homomorphism h such that $G_2[\bar{r}/x]G$, and where h is defined as $h(1'') = y$, $h(1') = \epsilon$ and $h(n') = z$, where $1''$, $1'$, n', y, and z are as given before. Let G' be G_2, then the proof is complete. ∎

Notice that with the cover-homomorphism of Theorem 4.2 we have both $G_2[l/x]G$ and $G_2[\bar{r}/x]G$. This is not difficult to understand since the productions which support the cover appear in both left and reversed right parses in the same order.

Up to this point we have been concerned with context-free grammars G such that $L(G)$ is prefix-free. In the following corollary which concludes this section we consider both prefix-free and non-prefix-free languages. The first part of this corollary is in fact Theorem 4.2. For the proof of the second part we need the definition of an $LR(1)$ grammar as presented in Section 1.

COROLLARY 4.1. *Let G be a parsable grammar with a valid* dpdt *which produces x-parses. Then*

(i) *if $L(G)$ is prefix-free, there exists a strict deterministic grammar G' such that $G'[\bar{r}/x]G$;*

(ii) *otherwise there exists an $LR(1)$ grammar G' such that $G'[\bar{r}/x]G$.*

Proof of the second part. Suppose $G = (N, \Sigma, P, S)$ is an x-parsable grammar and $L(G)$ is not necessarily prefix-free. Define cfg $G_1 = (N \cup \{S_1\}, \Sigma \cup \{\perp\}, P \cup \{S_1 \to S\perp\}, S_1\}$, where S_1 is not already in N and $S_1 \to S\perp$ is a production with label 0. Clearly G_1 is x-parsable and there exists a strict deterministic grammar $G_2 = (N_2, \Sigma_2, P_2, S_2)$, where $\Sigma_2 = \Sigma \cup \{\perp\}$, such that $G_2[\bar{r}/x]\, G_1$ with cover-homomorphism h. G_2 has type-1 productions (see the proof of Theorem 4.2) with $\perp$ on the right-hand side. These productions are of the form $A \to \perp B_1 B_2 \cdots B_n$, $n \geqslant 0$ and $A, B_1, B_2, \ldots, B_n \in N_2$. Let Q be the subset of P_2 which contains only such productions.

Define $R = \{A \to B_1 B_2 \cdots B_n \mid A \to \perp B_1 B_2 \cdots B_n \in Q\}$, where each production in R has the same label as the corresponding production in Q, and let R' be the complement of Q in P_2. Define $G' = (N', \Sigma', P', S')$, where $N' = N_2$, $\Sigma' = \Sigma$, $P' = R \cup R'$, and $S' = S_2$. Clearly $L(G') = L(G)$ (necessarily we have for a production $A \to \perp B_1 B_2 \cdots B_n \in Q$ that $B_1 B_2 \cdots B_n \overset{*}{\Rightarrow} \epsilon$) and $G'[\bar{r}/x]G$ with a cover-homomorphism h' which is defined

$$\begin{aligned} h'(p) &= \epsilon && \text{for each } p \in P' \text{ such that } h(p) = 0, \\ h'(p) &= h(p) && \text{otherwise.} \end{aligned}$$

We prove G' is an $LR(1)$ grammar. Suppose G' is not $LR(1)$, then there exist γ, α, α', $\beta, \beta' \in (N' \cup \Sigma')^*$; $w, w', x \in \Sigma'^*$; $A, A' \in N'$ such that

(1) $S' \overset{*}{\Rightarrow}_r \alpha A w \Rightarrow_r \alpha\beta w = \gamma w$

and

(2) $S' \overset{*}{\Rightarrow}_r \alpha' A' x \Rightarrow_r \alpha'\beta' x = \gamma w'$

and

(3) $1 : w = 1 : w'$ and $(A \to \beta, \mid \alpha\beta \mid) \neq (A' \to \beta', \mid \alpha'\beta' \mid)$.

We prove that this contradicts the property that G_2 is strict deterministic and hence an $LR(0)$ grammar. Notice that the only difference between G' and G_2 is in the productions in R and Q. For (1) and (2) we have corresponding derivations (1') and (2') in G_2, that is, if in (1) or (2) a production of R is used then in (1') and (2') the corresponding production of Q is used and otherwise the same productions are used.

We distinguish between two cases.

Case 1. Assume $1 : w = \epsilon$, then $w = w' = \epsilon$ and G' is ambiguous. The corresponding derivations for (1) and (2) can be written as

(1') $S' \overset{*}{\Rightarrow}_r \gamma'$ and (2') $S' \overset{*}{\Rightarrow}_r \gamma''$

where

$\gamma' = \gamma_1 \perp \gamma_2$, with $\gamma_1\gamma_2 = \gamma$, if a production of R is used in (1),
$\gamma' = \gamma$, if no production of R is used in (1),
$\gamma'' = \gamma_3 \perp \gamma_4$, with $\gamma_3\gamma_4 = \gamma$, if a production of R is used in (2),
$\gamma'' = \gamma$, if no production of R is used in (2).

It is sufficient to consider the following three cases.

a. The case $\gamma' = \gamma'' = \gamma$ contradicts G_2 is unambiguous.

b. Let $\gamma' = \gamma_1 \perp \gamma_2$ and $\gamma'' = \gamma = \gamma_1\gamma_2$. Then, with $v \in L(\gamma_1)$, we have that both v and $v\perp$ in $L(G_2)$ which is impossible.

c. Let $\gamma' = \gamma_1 \perp \gamma_2$ and $\gamma'' = \gamma_3 \perp \gamma_4$. Necessarily we have again both $\gamma_2 \overset{*}{\Rightarrow} \epsilon$ and $\gamma_4 \overset{*}{\Rightarrow} \epsilon$. If $\gamma_1 = \gamma_3$ then one can easily verify, by considering the different ways $\perp$ can be introduced, that G_2 is ambiguous. Moreover, if $\gamma_1 \neq \gamma_3$ then, if $v \in L(\gamma)$, we have both $v \in L(\gamma_1)$ and $v \in L(\gamma_2)$ which leads again to the false conclusion that G_2 is ambiguous.

Case 2. Assume $1 : w \in \Sigma$, then the corresponding derivations for G_2 are

(1') $S' \overset{*}{\Rightarrow}_r \alpha A w \perp \Rightarrow_r \alpha\beta w \perp = \gamma w \perp$,

and

(2a') $S' \overset{*}{\Rightarrow}_r \alpha' A' x \perp \Rightarrow_r \alpha'\beta' x \perp = \gamma w' \perp$,

or

(2b') $S' \overset{*}{\Rightarrow}_r \alpha' A' \perp \Rightarrow_r \alpha' \perp = \gamma w' \perp$.

Since $1 : w = 1 : w'$ and $(A \to \beta, \mid \alpha\beta \mid) \neq (A' \to \beta', \mid \alpha'\beta' \mid)$ we have both the existence of (1′) and (2a′) and of (1′) and (2b′) contradict the fact that G_2 is $LR(0)$.

This completes the proof that G' is $LR(1)$. ∎

5. Conclusions

The importance of Theorem 4.2 is that every cfg for which a parsing method exists that can be implemented by a dpdt is covered by a strict deterministic grammar. Since the cover is such that reversed right parses of the strict deterministic grammar are considered parsing the strict deterministic grammar is as good as parsing the original grammar. For $x = \bar{r}$ we see that every reversed right parsable grammar is right-covered (see Definition 2.1) by a strict deterministic grammar. According to the inclusion diagram of Section 3 and our assumption of prefix-free languages we can conclude that every $LR(k)$ grammar is right-covered by a strict deterministic grammar. According to the remark following Definition 1.2 we can conclude that every $LR(k)$ grammar is right-covered by an $LR(0)$ grammar (or without the assumption of prefix-free languages by an $LR(1)$ grammar).

Acknowledgments

I am grateful to Professor L. A. M. Verbeek for some helpful comments. I also acknowledge the referee for his useful remarks.

References

1. A. V. Aho and J. D. Ullman, "The Theory of Parsing, Translation and Compiling," Vols. 1 and 2, Prentice–Hall, Englewood Cliffs, N. J., 1972 and 1973.
2. M. M. Geller and M. A. Harrison, On LR(k) grammars and languages, manuscript, 1975.
3. J. Gray and M. A. Harrison, On the covering and reduction problems for context-free grammars, *J. Assoc. Comput. Mach.* **19** (1972), 675–698.
4. M. Hammer, A new grammatical transformation into LL(k) form, *in* "Conf. Record of the 6th Ann. ACM Sympos. on Theory of Computing, 1974," pp. 266–275.
5. M. A. Harrison, On covers and precedence analysis, *in* "Lecture Notes in Computer Science 1, G.I. 3. Jahrestagung, 8–10 Okt. 1973," pp. 2–17.
6. M. A. Harrison and I. M. Havel, Strict deterministic grammars, *J. Comput. System Sci.* **7** (1973), 237–277.
7. M. A. Harrison and I. M. Havel, On the parsing of deterministic languages, *J. Assoc. Comput. Mach.* **21** (1974), 525–548.
8. M. D. Mickunas, On the complete covering problem for LR(k) grammars, *J. Assoc. Comput. Mach.* **23** (1976), 17–30.
9. M. D. Mickunas, R. L. Lancaster, and V. B. Schneider, Transforming LR(k) grammars to LR(1), SLR(1), and (1,1) bounded right-context grammars, *J. Assoc. Comput. Mach.* **23** (1976), 511–533.

10. M. D. MICKUNAS AND V. B. SCHNEIDER, On the ability to cover LR(k) grammars with LR(1), SLR(1) and (1,1) bounded-context grammars, *in* "14th Ann. Sympos. on Sw. and Aut. Theory, IEEE 1973," pp. 109–121.
11. A. NIJHOLT, "On the Covering of Parsable Grammars, TW-Memorandum No. 96, 1975," Twente University of Technology.
12. D. J. ROSENKRANTZ AND P. M. LEWIS, Deterministic left corner parsing, *in* "11th Ann. Sympos. on Sw. and Aut. Theory, IEEE 1970," pp. 139–152.

Printed by the St Catherine Press Ltd., Tempelhof 37, Bruges, Belgium