

# Transforming $LR(k)$ Grammars to $LR(1)$ , $SLR(1)$ , and $(1,1)$ Bounded Right-Context Grammars

M. D. MICKUNAS

*University of Illinois at Urbana-Champaign, Urbana, Illinois*

R. L. LANCASTER

*Bowling Green State University, Bowling Green, Ohio*

AND

V. B. SCHNEIDER

*Purdue University, West Lafayette, Indiana*

**ABSTRACT.** A method is presented for directly transforming an arbitrary  $LR(k)$  grammar to an equivalent  $LR(1)$  grammar. It is further shown that the method transforms an arbitrary prefix-free  $LR(k)$  grammar to an equivalent  $LR(0)$  grammar. It is argued that the method is efficient and offers some advantages over traditional "look-ahead" parsing methods. Finally, it is demonstrated that the method can be used to transform an  $LR(1)$  grammar to an equivalent  $SLR(1)$  grammar, which in turn can be easily transformed to an equivalent  $(1, 1)$  bounded right-context grammar.

**KEY WORDS AND PHRASES:** parsing, context-free grammars,  $LR(k)$  grammars, grammatical transformations

**CR CATEGORIES:** 4.12, 5.22, 5.23

## 1. Introduction

Whenever a parsing technique is discussed in the literature, the author usually generalizes the method to permit "look-ahead" calculations (i.e. the parser, when visualized as a pushdown automaton, is permitted to look beyond the current symbol of its input, in order to determine its next move). Thus an  $LR(1)$  method [16] is usually augmented by including the calculation of " $k$ -symbol input sets" (on an exception basis) instead of only "1-symbol input sets," whence the method is generalized to  $LR(k)$  [3, 4, 16, 17]. A precedence method [25] is usually augmented by permitting duplicate right-parts in production rules. The grammar remains precedence detectable [11] but the reduction phase of the precedence parser must include a " $k$ -symbol look-ahead" ability (again on an exception basis) to choose properly between duplicate right-parts of rules. Precedence methods may thus be extended to handle  $(1, k)$  bounded right-context grammars

Copyright © 1976, Association for Computing Machinery, Inc. General permission to republish, but not for profit, all or part of this material is granted provided that ACM's copyright notice is given and that reference is made to the publication, to its date of issue, and to the fact that reprinting privileges were granted by permission of the Association for Computing Machinery.

This work was supported, in part, by National Science Foundation Grant GJ-851 to Purdue University and National Science Foundation Grant DCR 72-03740 A01 to the University of Illinois at Urbana-Champaign.

Authors' addresses M.D. Mickunas, Department of Computer Science, Digital Computer Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801; R.L. Lancaster, Computer Science Department, Bowling Green State University, Bowling Green, OH 43402; V.B. Schneider, Department of Computer Sciences, Mathematical Sciences Building, Purdue University, West Lafayette, IN 47907.

[1, 6, 9, 11]. The fact that look-ahead is performed on only an exception basis is a consequence of two disadvantages of the technique: (1) look-ahead sets are difficult to calculate, and (2) provision of  $k$ -symbol look-ahead sets for those portions of the parser that don't need it consumes an enormous amount of memory space.

In principle, the technique of transforming a grammar to LR(1) overcomes the above two disadvantages. Moreover, since there are fewer parsing actions to be performed, the parser lends itself to simpler analyses of its space/time requirements. But there are two major objections to the policy of transforming a grammar to LR(1): (1) the structure imposed by the original grammar is generally not preserved, and (2) the transformation may exorbitantly (sometimes exponentially) increase the number of production rules.

From a compiler-writer's point of view, the first objection is often academic. The practitioner is concerned not with the changes in the grammatical structure per se, but with how those changes affect the code-generating (semantic) properties of the compiler. The transformation techniques presented in this paper permit a compiler's semantics to adjust to such changes in grammatical structure in three ways. To illustrate these methods, consider the following stylized subset of an Algol-like grammar:

$$\begin{aligned}\pi_1 &: \langle \text{statement} \rangle \rightarrow \langle \text{assignment} \rangle \\ \pi_2 &: \langle \text{statement} \rangle \rightarrow \langle \text{label} \rangle : \langle \text{statement} \rangle \\ \pi_3 &: \langle \text{label} \rangle \rightarrow \textit{identifier} \\ \pi_4 &: \langle \text{assignment} \rangle \rightarrow \langle \text{variable} \rangle : = \langle \text{expression} \rangle \\ \pi_5 &: \langle \text{variable} \rangle \rightarrow \textit{identifier}\end{aligned}$$

The above grammar is LR(2). The semantics of  $\pi_3$  would attribute the meaning of "label" to an *identifier*, whereas the semantics of  $\pi_5$  would attribute the meaning of "variable" to it. In each case, the *identifier* (or its encoding) is located by the code-generating routine as the last symbol which was scanned by the parser.

A non-LR(1) problem arises with the rules  $\pi_3$  and  $\pi_5$ . To reduce an *identifier*, an LR(2) parser will apply  $\pi_5$  whenever the *identifier* is followed by  $: =$ , and it will apply  $\pi_3$  whenever the *identifier* is followed by  $: \textit{identifier}$ . For example,

$$\textit{identifier} : \textit{identifier} : = \dots$$

should be parsed by applying the rules  $\pi_3, \pi_5, \dots, \pi_4, \pi_1, \pi_2$  yielding

$$\begin{array}{ll}\text{then} & \langle \text{label} \rangle : \textit{identifier} : = \dots \\ \vdots & \vdots \\ \text{then} & \langle \text{label} \rangle : \langle \text{assignment} \rangle \dots \\ \text{then} & \langle \text{label} \rangle : \langle \text{statement} \rangle \dots \\ \text{then} & \langle \text{statement} \rangle \dots\end{array}$$

The transformations we will present would transform the above grammar to the LR(1) grammar:

$$\begin{aligned}\pi'_1 &: \langle \text{statment} \rangle \rightarrow \langle \text{assignment} \rangle \\ \pi'_2 &: \langle \text{statement} \rangle \rightarrow \langle \text{labeler} \rangle \langle \text{statement} \rangle \\ \pi'_3 &: \langle \text{labeler} \rangle \rightarrow \textit{identifier} : \\ \pi'_4 &: \langle \text{assignment} \rangle \rightarrow \langle \text{left-part} \rangle = \langle \text{expression} \rangle \\ \pi'_5 &: \langle \text{left-part} \rangle \rightarrow \textit{identifier} :\end{aligned}$$

If the original grammar contained code-generating subroutines associated with rules  $\pi_3$  or  $\pi_5$ , then some alteration of these subroutines may be necessary to work correctly with rules  $\pi'_3$  and  $\pi'_5$  of the altered grammar. The semantics of  $\pi_3$  ( $\pi_5$ ) must still supply the "label" ("variable") meaning to the *identifier*. However, the *identifier* (or its encoding) is now located by the code-generating routine as the next-to-last symbol which was scanned by the parser. A second technique allows the semantics to remain unchanged. The transformed grammar is used to parse the program "internally" and, as the internal parse proceeds, an "external" parse via the original grammar is induced. As

this external parse proceeds, the original semantics can be applied. This technique was first suggested by Gray and Harrison [11] and arises from the notion of "grammatical covers," originally due to Reynolds and Haskell [23]. In order to accomplish the internal/external linkage, we map the new rules onto the old rules by a function,  $\phi$ , which in this example simply maps each  $\pi_i'$  to  $\pi_i$ . While we will not present a formal proof that this technique works, we will nonetheless have further detailed remarks to make about it. Schematically, the internal/external parses proceed as follows:

internally	externally
$identifier : identifier : = \dots$	$identifier : identifier : = \dots$
apply $\pi_3'$	apply $\phi(\pi_3') = \pi_3$
$\langle \text{labeler} \rangle identifier : = \dots$	$\langle \text{label} \rangle : identifier : = \dots$
apply $\pi_5'$	apply $\phi(\pi_5') = \pi_5$
$\langle \text{labeler} \rangle \langle \text{left-part} \rangle = \dots$	$\langle \text{label} \rangle : \langle \text{variable} \rangle : = \dots$
$\vdots$	$\vdots$
apply $\pi_4'$	apply $\phi(\pi_4') = \pi_4$
$\langle \text{labeler} \rangle \langle \text{assignment} \rangle \dots$	$\langle \text{label} \rangle : \langle \text{assignment} \rangle \dots$
apply $\pi_1'$	apply $\phi(\pi_1') = \pi_1$
$\langle \text{labeler} \rangle \langle \text{statement} \rangle \dots$	$\langle \text{label} \rangle : \langle \text{statement} \rangle \dots$
apply $\pi_2'$	apply $\phi(\pi_2') = \pi_2$
$\langle \text{statement} \rangle \dots$	$\langle \text{statement} \rangle \dots$

A third way of adapting the grammar to its semantics in this case is to force the lexical scanner (which supplies an encoded form of the input to the parser) to recognize the sequence of symbols  $: =$  as a *single* input symbol ( $:=$ ). Thus, the grammar may be changed to

- $\pi_1 : \langle \text{statement} \rangle \rightarrow \langle \text{assignment} \rangle$
- $\pi_2 : \langle \text{statement} \rangle \rightarrow \langle \text{label} \rangle : \langle \text{statement} \rangle$
- $\pi_3 : \langle \text{label} \rangle \rightarrow identifier$
- $\pi_4 : \langle \text{assignment} \rangle \rightarrow \langle \text{variable} \rangle := \langle \text{expression} \rangle$
- $\pi_5 : \langle \text{variable} \rangle \rightarrow identifier$

The parser can now decide to apply  $\pi_3$  if the next input symbol is  $:$  and apply  $\pi_5$  if it is  $:=$ . This is a standard technique that is usually applied in an ad hoc fashion.

In our experience, the transformation to LR(1) does not exorbitantly increase the size of the grammar, so long as the *language* in question is well suited to LR(1) parsing. We have found that, even if the original grammar is badly written, the transformation will produce a grammar of reasonable size. Even for pathological languages, the transformation produces LR(1) grammars which are competitive with handwritten LR(1) grammars for the same language. In the remainder of this section we present basic definitions and notation. Most of what follows is standard, and we have patterned much of the format after that of Gray and Harrison [11].

*Definition.* A (context-free) grammar (CFG) is a four-tuple,  $G = (V, \Sigma, P, S)$  where

- (a)  $V$  is a finite nonempty set of symbols (*vocabulary*);
- (b)  $\Sigma \subset V$  (*terminal vocabulary*);
- (c)  $N = V - \Sigma$  (*nonterminal vocabulary*);
- (d)  $S \in N$  (*goal symbol*); and
- (e)  $P$  is a finite subset<sup>1</sup> of  $N \times V^*$  (*productions*).

We will denote an element  $(A, v)$  of  $P$  as  $A \rightarrow v$ , and we will often ascribe indices to productions:  $\pi_i = A \rightarrow v$ . For any rule,  $A \rightarrow v$ ,  $A$  is called the *left-part* and  $v$  the *right-part* of the rule. Any rule having left-part  $A$  is called an  $A$ -rule.

<sup>1</sup> From [11] Let  $X$  and  $Y$  be sets of words. We write  $XY = \{xy \mid x \in X, y \in Y\}$  where  $xy$  is the concatenation of  $x$  and  $y$ . Define  $X^0 = \{\Lambda\}$  where  $\Lambda$  is the null word. For each  $i \geq 0$  define  $X^{i+1} = X \cdot X^i$  and  $X^* = \bigcup_{i \geq 0} X^i$ . Let  $X^+ = X^*X$  and let  $\emptyset$  denote the empty set.

We find it convenient to employ binary relations on sets of words.

**Definition** (from [11]). Let  $\rho$  be a binary relation on a set  $X$ , i.e.  $\rho \subseteq X \times X$ . Define  $\rho^0 = \{(a, a) \mid a \in X\}$ , and for each  $i \geq 0$ ,  $\rho^{i+1} = \rho^i \rho$ . Last,  $\rho^* = \bigcup_{i \geq 0} \rho^i$  and  $\rho^+ = \rho^* \rho$ . For a binary relation  $\rho$  on  $X$ ,  $\rho^*$  is the *reflexive-transitive closure* of  $\rho$  while  $\rho^+$  is the *transitive closure* of  $\rho$ . For  $a \in X$  we will write  $a\rho$  to denote  $\{b \in X \mid a\rho b\}$  and similarly  $\rho a$  to denote  $\{b \in X \mid b\rho a\}$ . For  $H \subseteq X$  we write  $H\rho$  to denote  $\{b \in X \mid (\text{there exist } a \in H) [a\rho b]\}$  and similarly  $\rho H$  to denote  $\{b \in X \mid (\text{there exist } a \in H) [b\rho a]\}$ . We will also write  $a_1, a_2, \dots, a_n \rho H$  instead of  $a_1, a_2, \dots, a_n \in \rho H$  or  $\{a_1, a_2, \dots, a_n\} \subseteq \rho H$ .

We employ the usual binary relation  $\Rightarrow \subseteq V^* \times V^*$ , writing  $u \Rightarrow v$  instead of  $(u, v) \in \Rightarrow$ .

**Definition.** Let  $G = (V, \Sigma, P, S)$  ( $N = V - \Sigma$ ) be a CFG and let  $u, v \in V^*$ . Define  $u \Rightarrow v$  if there exist  $x, w \in V^*$ ;  $y \in \Sigma^*$ ;  $A \in N$  for which  $u = xAy$ ,  $v = xwy$ , and  $A \rightarrow w$  is in  $P$ . Furthermore, we write the reflexive-transitive closure of  $\Rightarrow$  as  $\Rightarrow^*$ . If we wish to make clear that the grammar  $G$  is being used, we will write  $\Rightarrow^G$ .

**Note** By this definition we have immediately restricted our attention to *rightmost*, or *canonical* derivations

**Definition.** The set of (*canonical*) *sentential forms* for a CFG,  $G = (V, \Sigma, P, S)$ , is denoted by  $\text{CSF}(G)$  and is defined as  $\text{CSF}(G) = \{x \in V^* \mid S \Rightarrow^* x\}$ . The *language* generated by  $G$  is denoted by  $L(G)$  and is defined as  $L(G) = \Sigma^* \cap \text{CSF}(G)$ .

**Definition.** Let  $G = (V, \Sigma, P, S)$  be a CFG and let  $x_i \in V^*$ , ( $0 \leq i \leq r$ ). If  $x_i \Rightarrow x_{i+1}$  by applying the production  $\pi_i \in P$ , then we say that  $x_i$  *directly derives*  $x_{i+1}$  *via*  $\pi_i$ . If  $x_0 \Rightarrow x_i \Rightarrow \dots \Rightarrow x_r$ , where  $x_i$  directly derives  $x_{i+1}$  via  $\pi_i$ , ( $0 \leq i < r$ ), then we say that  $x_0$  (*canonically*) *derives*  $x_r$  *via*  $(\pi_i)_{i=0}^{r-1}$  and that  $(\pi_i)_{i=0}^{r-1}$  is a (*canonical*) *derivation* of  $x_r$  from  $x_0$ .

**Definition.** A CFG  $G$  is said to be *unambiguous* if and only if every  $x \in L(G)$  has exactly one canonical derivation. A CFG which is not unambiguous is said to be *ambiguous*.

**Definition.** Let  $G = (V, \Sigma, P, S)$  ( $N = V - \Sigma$ ) be a CFG and let  $u \in V^*$ ,  $A \in N$ ,  $w \in \Sigma^*$  such that  $uAw \in \text{CSF}(G)$ . Let  $\pi = A \rightarrow v$  in  $P$ . Then  $uvw \in \text{CSF}(G)$  is said to have<sup>3</sup> *handle*  $(\pi, |uw|)$ . By convention,  $S \in \text{CSF}(G)$  has handle<sup>4</sup>  $(\Lambda, 1)$ .

**Definition.** Let  $k$  be a nonnegative integer. A CFG,  $G = (V, \Sigma, P, S)$ , is said to be  $\text{LR}(k)$  if and only if<sup>5</sup> for all  $x \in V^*$ ;  $y, y' \in \Sigma^*$ ;  $\pi, \pi' \in P_A$ ; if  $xy \in \text{CSF}(G)$  has handle  $(\pi, |x|)$  and  $xy' \in \text{CSF}(G)$  has handle  $(\pi', |j|)$  and  $^{(k)}y = ^{(k)}y'$  then<sup>6</sup>  $(\pi, |x|) = (\pi', |j|)$ . Otherwise  $G$  is said to be non- $\text{LR}(k)$ , and in that case any rule,  $\pi$ , which violates the above conditions is said to be non- $\text{LR}(k)$ . It is well known that an  $\text{LR}(k)$  grammar is unambiguous.

**Definition.** A CFG,  $G = (V, \Sigma, P, S)$  ( $N = V - \Sigma$ ), is said to be

(a)  $\Lambda$ -free if  $P \subseteq N \times V^+$ ,

(b)  $\Lambda$ -isolated if either (i)  $G$  is  $\Lambda$ -free, or (ii)  $P \subseteq (N \times (V - \{S\}))^+ \cup \{S \rightarrow \Lambda\}$ ,

(c) reduced if for each  $A \in V - \{S\}$ , (i) there exist  $x, y \in V^*$  for which  $xAy \in \text{CSF}(G)$ , (ii) there exists  $z \in \Sigma^*$  for which  $A \Rightarrow^* z$ .

<sup>2</sup> From [11]. The operation is a *composition* of relations which is defined as follows. if  $\rho \subseteq X \times Y$  and  $\sigma \subseteq Y \times Z$ , define  $\rho\sigma = \{(x, z) \mid (x, y) \in \rho \text{ and } (y, z) \in \sigma \text{ for some } y \in Y\}$ . Observe that  $\rho\sigma \subseteq X \times Z$ .

<sup>3</sup> Let  $x = a_1 a_2 \dots a_n \in V^*$  for some  $n \geq 0$  (where each  $a_i \in V$  and where, if  $n = 0$ ,  $x = \Lambda$ ). The *length* of  $x$  is denoted by  $|x|$  and is defined as  $|x| = n$ . For  $i < n$ ,  $^{(i)}x = a_1 \dots a_i$ , and  $x^{(i)} = a_{n-i+1} \dots a_n$ . For  $i \geq n$ ,  $^{(i)}x = x^{(i)} = x$ .

<sup>4</sup>  $\Lambda$  is also used to denote the null production

<sup>5</sup> If  $X$  is a set, then  $X_\Lambda = X \cup \{\Lambda\}$ .

<sup>6</sup> Notice that as a result of our convention that  $S \in \text{CSF}(G)$  has handle  $(\Lambda, 1)$ , we avoid the "ambiguous  $\text{LR}(0)$  grammar" anomaly cited by Harrison [12]. Our definition is equivalent to that of the "Augmented  $\text{LR}(k)$ " grammars of Geller and Harrison [7]. Thus, for example, the grammar with productions  $S \rightarrow Sa \mid a$  is not  $\text{LR}(0)$  by our definition since  $S \in \text{CSF}(G)$  has handle  $(\Lambda, 1)$ ,  $Sa \in \text{CSF}(G)$  has handle  $(S \rightarrow Sa, 2)$ , and the handles are not the same.

There are some additional useful relations which we shall employ.

*Definition.* Let  $G = (V, \Sigma, P, S)$  ( $N = V - \Sigma$ ) be a CFG, and let  $X, Y \in V$ .

(a)  $X \lambda Y$  ( $X$  left-derives  $Y$ ) if and only if there exists  $u \in V^*$  for which  $X \rightarrow Yu$  is in  $P$ .

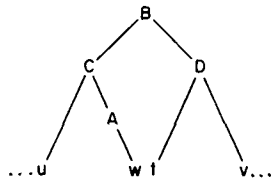
(b)  $X \rho Y$  ( $X$  right-derives  $Y$ ) if and only if there exists  $u \in V^*$  for which  $X \rightarrow uY$  is in  $P$ .

(c)  $X \tau Y$  ( $X$  left-derives terminal  $Y$ ) if and only if  $X \lambda^* Y$  and  $Y \in \Sigma$ .

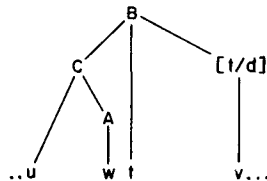
(Thus  $X \lambda^* Y$  is an abbreviation for “(there exists  $u \in V^*$ ) [ $X \Rightarrow^* Yu$ ]” and  $X \rho^* Y$  for “(there exists  $u \in V^*$ ) [ $X \Rightarrow^* uY$ ]”. We will often be presented with a set  $H \subseteq V$  and will wish to identify an element,  $A$ , of  $\{X \in V \mid (\text{there exist } Y \in H, u \in V^*) [X \Rightarrow^* uY]\}$ . Using the relation  $\rho$ , we will constrain such an element by simply specifying  $A \rho^* H$ .)

## 2. Preliminary Remarks

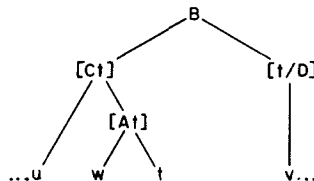
The main idea behind conversion from LR( $k$ ) to LR(1) can be illustrated as follows: Suppose a rule,  $A \rightarrow w$ , participates in an LR(2) (non-LR(1)) portion of a grammar. The decision of whether to apply the reduction  $A \rightarrow w$  can be made on the basis of two symbols of look-ahead. If  $A \rightarrow w$  is indeed the correct reduction for a sentential form  $\dots u w t v \dots$  then the next input symbol,  $t$ , must eventually fit into the parse, perhaps via a structure like



The conversion scheme must alter the offending rule  $A \rightarrow w$  so that the decision to apply it is not forced until after the symbol  $t$  has been scanned. The technique we use is to “extract” the right-context first:



and then to merge the extracted branch with its left neighbor so as to force a premature scanning of the extracted symbol



The algorithm that we will develop performs such extraction upon all right-contexts for the symbol  $A$  of the offending rule  $A \rightarrow w$  and performs a premature scan merger on all such right-contexts for all  $C$  for which  $C \rho^* A$ . We consider a few more examples.

The LR (2) grammar

$S \rightarrow Abb \mid Bbc$

$A \rightarrow aA \mid a$

$B \rightarrow aB \mid a$

with offending rules  $A \rightarrow a$  and  $B \rightarrow a$  requires no right-context extraction, and is converted via premature scanning to the LR(1) grammar

$$\begin{aligned} S &\rightarrow [Ab]b \mid [Bb]c \\ [Ab] &\rightarrow a[Ab] \mid ab \\ [Bb] &\rightarrow a[Bb] \mid ab \end{aligned}$$

The LR(2) grammar

$$\begin{aligned} S &\rightarrow Abb \mid Bbc \\ A &\rightarrow Aa \mid a \\ B &\rightarrow a \end{aligned}$$

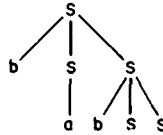
with offending rules  $A \rightarrow a$  and  $B \rightarrow a$  requires no right-context extraction, and is converted via premature scanning to the LR(1) grammar

$$\begin{aligned} S &\rightarrow [Ab]b \mid [Bb]c \\ [Aa] &\rightarrow [Aa]a \mid aa \\ [Ab] &\rightarrow [Aa]b \mid ab \\ [Bb] &\rightarrow ab \end{aligned}$$

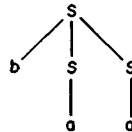
Finally, the LR(2) grammar

$$S \rightarrow bSS \mid a \mid aac$$

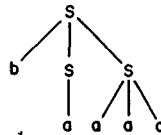
has the offending rule  $S \rightarrow a$ , whose right contexts are generated by the second  $S$  in the right-part of  $S \rightarrow bSS$ ; those right-contexts are  $a$  and  $b$ ;



and



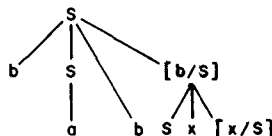
and



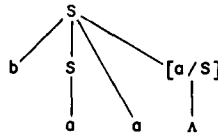
The grammar is converted first (under right-context extraction) to

$$\begin{aligned} S &\rightarrow bSa[a/S] \mid bSb[b/S] \mid a \mid aac \\ [a/S] &\rightarrow \Lambda \mid ac \\ [b/S] &\rightarrow Sa[a/S] \mid Sb[b/S]. \end{aligned}$$

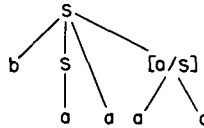
The right-contexts (whether  $a$  or  $b$ ) for  $S \rightarrow a$  have thus been extracted in all situations:



( $x = a$  or  $b$ ) and



and



Moreover, the new rule,  $[a/S] \rightarrow \Lambda$ , which is also non-LR(1), has had its right-contexts extracted. Finally, the grammar is converted (under premature scanning) to the LR(1) grammar

$$\begin{aligned}
 S &\rightarrow b[Sa][a/S] \mid b[Sb][b/S] \mid a \mid aac \\
 [Sa] &\rightarrow b[Sa][[a/S]a] \mid b[Sb][[b/S]a] \mid aa \mid aaca \\
 [Sb] &\rightarrow b[Sa][[a/S]b] \mid b[Sb][[b/S]b] \mid ab \mid aacb \\
 [a/S] &\rightarrow \Lambda \mid ac \\
 [[a/S]a] &\rightarrow a \mid aca \\
 [[a/S]b] &\rightarrow b \mid acb \\
 [b/S] &\rightarrow [Sa][a/S] \mid [Sb][b/S] \\
 [[b/S]a] &\rightarrow [Sa][[a/S]a] \mid [Sb][[b/S]a] \\
 [[b/S]b] &\rightarrow [Sa][[a/S]b] \mid [Sb][[b/S]b]
 \end{aligned}$$

This transformation can be optimized to prematurely scan only the conflicting right-contexts for the offending rule. In this case, although both  $a$  and  $b$  have been extracted, only  $a$  is a conflicting right-context. Thus, after right-context extraction, we can obtain instead the following LR(1) grammar:

$$\begin{aligned}
 S &\rightarrow b[Sa][a/S] \mid b[Sb][b/S] \mid a \mid aac \\
 [Sa] &\rightarrow b[Sa][[a/S]a] \mid b[Sb][[b/S]a] \mid aa \mid aaca \\
 [a/S] &\rightarrow \Lambda \mid ac \\
 [[a/S]a] &\rightarrow a \mid aca \\
 [b/S] &\rightarrow [Sa][a/S] \mid Sb[b/S] \\
 [[b/S]a] &\rightarrow [Sa][[a/S]a] \mid Sb[[b/S]a]
 \end{aligned}$$

This grammar contains 16 rules. So far as we can tell, every LR(1) grammar for this language must contain at least 12 rules. Thus, although the size of the grammar has increased dramatically, the increase is due to the *language* involved, and is not ascribable to any wholesale inefficiencies in the transformations.

### 3. Main Results

We now turn to the development of our primary result, a procedure for reducing lookahead. The procedure is presented as an iterative application of a succession of transformations on grammars. Besides the Right-Context Extraction and Premature Scanning transformations, we use three utility transformations which serve, for the most part, to simplify operations. The first of these utility transformations is the Right-Stratification Transformation, which simply breaks a single production rule,  $A \rightarrow uv$ , into two rules,  $A \rightarrow u[v]$  and  $[v] \rightarrow v$ .

*Right-Stratification Transformation.* Let  $G = (V, \Sigma, P, S)$  be a CFG. To right-stratify a rule  $A \rightarrow w$  in  $P$  at the  $u, v$  interface, we define

$$P_1 = \{A \rightarrow w\}, \quad P_2 = P - P_1.$$

We then define a new grammar,  $G''$ , which differs from  $G$  in its nonterminal vocabulary and its set of productions.

$$\begin{aligned} P_1'' &= \{A \rightarrow u[v] \mid A \rightarrow uv \text{ is in } P_1\}, & P'' &= P_1'' \cup P_2'' \cup P_3'', \\ P_2'' &= \{[v] \rightarrow v \mid A \rightarrow uv \text{ is in } P_1\}, & V'' &= V \cup \{[v]\}. \\ P_3'' &= P_2, \end{aligned}$$

LEMMA 1. Let  $G'' = (V'', \Sigma, P'', S)$  ( $N'' = V'' - \Sigma$ ) be obtained from  $G = (V, \Sigma, P, S)$  ( $N = V - \Sigma$ ) by means of the Right-Stratification Transformation. Then

- (a)  $L(G'') = L(G)$ ; and
- (b) there is a surjection,<sup>7</sup>  $\phi$ , from  $P_\Delta''$  onto  $P_\Delta$  such that if  $\pi \in P_\Delta''$  is not  $LR(j)$  then  $\phi(\pi) \in P_\Delta$  is not  $LR(j)$ .

PROOF. Right-stratification is a common transformation and is used by both McAfee and Presser [18] and Graham [9]. McAfee and Presser present an informal proof that  $L(G'') = L(G)$ . Graham gives a complete proof that  $L(G'') = L(G)$  as well as a proof that the right-context bound is preserved. By defining  $\phi$  as the identity on  $P_3''$  (thus mapping  $P_{3\Delta}''$  onto  $P_{2\Delta}$ ) and as a function from the singleton set  $P_1''$  onto  $P_1$  and from the singleton set  $P_2''$  onto  $P_1$ , we obtain the desired surjection. Embedding this in Graham's proof yields part (b) of the lemma.  $\square$

The Look-Ahead Reduction Procedure that we develop initially determines which rules of a grammar require right-context extraction and subsequent premature scanning of those right-contexts. This information is then transmitted in the set,  $H$ , of left-parts of the selected rules. The transformation we are about to present accomplishes the right-context extraction portion of the Look-Ahead Reduction Procedure.

It is necessary for the Right-Context Extraction Transformation to extract terminal symbols whenever they can occur as right contexts for one of the selected rules, i.e. if  $C \rightarrow v$  is a selected rule, and if some symbol,  $A$ , right-derives  $C$ , then whenever  $A$  appears in the right-part of a rule, it should either be rightmost or be followed by a terminal symbol. We formalize that objective by the following.

Definition. Let  $G = (V, \Sigma, P, S)$  ( $N = V - \Sigma$ ) be a CFG and let  $H \subseteq N$ .  $G$  is said to be in *H-right-context-extracted form* if and only if for every nonterminal  $A$  in  $\rho^*H$  and for every rule  $B \rightarrow uAy$  in  $P$ , either (a)  $y = \Lambda$ , or (b)  $y \in \Sigma V^*$ . Notice that this definition reduces to that of an operator form grammar [5, 11] in case  $H = N$  and  $G$  is reduced.

Notation. Let  $\pi = C \rightarrow v$  be a  $C$ -rule of  $P$ . If  $C$  is in  $H$  then  $\pi$  is also called an  $H$ -rule of  $P$ .

Right-Context Extraction Transformation. To extract the right-contexts for a reduced  $\Lambda$ -isolated grammar  $G = (V, \Sigma, P, S)$  ( $N = V - \Sigma$ ) (given  $H \subseteq N$ ) we first (iteratively) locate each rule

$$\pi = B \rightarrow uCvDEw \quad (E \in N; u, v, w \in V^*)$$

for which  $C$ ,  $D\rho^*H$ , and right-stratify  $\pi$  at the  $v, D$  interface (to obtain  $B \rightarrow uCv[DEw]$  and  $[DEw] \rightarrow DEw$ ). Although we might present the Right-Context Extraction Transformation without such stratification, the transformation would be not only more complex, but also less efficient (in general). This is because the transformation, given such a  $\pi$ , will accomplish an extraction of the right-contexts for both  $C$ -rules and  $D$ -rules (since  $C, D\rho^*H$ ). These contexts will then appear in modifications of  $\pi$ . In the unstratified case, the modifications must account for all possible pairs of such right-contexts, whereas with stratification, combinations of right-contexts are handled in series. For example, if  $v = Fx$  (whence  $\pi = B \rightarrow uCFxDEw$ ) and  $a, b$  are in  $F\tau$ ,  $c, d, e$  are in  $E\tau$ , then  $\pi$

<sup>7</sup> Let  $X$  and  $Y$  be sets, and  $\phi$  a mapping of  $X$  into  $Y$ . If each element of  $Y$  is the image of at least one element of  $X$ , then  $\phi$  is a surjection from  $X$  onto  $Y$ .

would yield

$$\begin{array}{ll} \pi_1 = B \rightarrow uCa[a/F]xDc[c/E]w & \pi_4 = B \rightarrow uCb[b/F]xDc[c/E]w \\ \pi_2 = B \rightarrow uCa[a/F]xDd[d/E]w & \pi_5 = B \rightarrow uCb[b/F]xDd[d/E]w \\ \pi_3 = B \rightarrow uCa[a/F]xDde[e/E]w & \pi_6 = B \rightarrow uCb[b/F]xDde[e/E]w \end{array}$$

However, by stratifying  $\pi$  as

$$\pi = B \rightarrow uCFx[DEw] \quad \pi' = [DEw] \rightarrow DEw$$

we obtain

$$\begin{array}{ll} \pi_1 = B \rightarrow uCa[a/F]x[DEw] & \pi_2' = [DEw] \rightarrow Dd[d/E]w \\ \pi_2 = B \rightarrow uCb[b/F]x[DEw] & \pi_3' = [DEw] \rightarrow De[e/E]w \\ \pi_1' = [DEw] \rightarrow Dc[c/E]w & \end{array}$$

Note that it is sometimes necessary to stratify a rule more than once. For example,

$$\pi = B \rightarrow xFuCuDEw \quad (E \in N),$$

where  $F, C, D\rho^*H$ , must be stratified as

$$\begin{array}{ll} \pi = B \rightarrow xFu[CvDEw] & \pi'' = [DEw] \rightarrow DEw \\ \pi' = [CvDEw] \rightarrow Cv[DEw] & \end{array}$$

Upon performing right-stratification, we have a (reduced  $\Lambda$ -isolated) grammar,  $G' = (V', \Sigma, P', S)$  ( $N' = V' - \Sigma$ ). We compute  $H' \subseteq N'$  such that for  $\pi$  in  $P'$ , if  $\phi(\pi)$  in  $P$  is an  $H$ -rule, then  $\pi$  is an  $H'$ -rule. (We can actually compute  $H'$  more carefully to yield a possibly smaller subset of  $N'$ , but the calculation given here will do.) To continue with the transformation, we then define

$$P_1' = \{A \rightarrow uBCv \text{ in } P' \mid C \in N'; B\rho^*H'\}$$

(Notice that as a result of stratification,  $C$  of such rules is uniquely determined.)

Let  $M' = \{C \in N' \mid A \rightarrow uBCv \text{ is in } P_1' \text{ where } B\rho^*H'\}$ . For each  $C \in M'$  define

$$\begin{array}{ll} Q'(C) = \{D \rightarrow Ew \text{ in } P' \mid E \in N'; C\lambda^*D\}, & P_3'(C) = Q'(C) - P_2'(C), \\ R'(C) = \{D \rightarrow aw \text{ in } P' \mid a \in \Sigma; C\lambda^*D\}, & P_4'(C) = R'(C) \cap P_1', \\ P_2'(C) = Q'(C) \cap P_1' & P_5'(C) = R'(C) - P_4'(C), \end{array}$$

and define

$$\begin{array}{l} P_i' = \bigcup_{C \in M'} P_i'(C) \quad (\text{for } i \in \{2, 3, 4, 5\}), \\ P_6' = P - P_1'. \end{array}$$

We then define a new grammar,  $G''$ , which differs from  $G'$  in its nonterminal vocabulary and its set of productions.

$$P_1'' = \{A \rightarrow uBa[a/C]v \mid C \in N'; C\tau a; A \rightarrow uBCv \text{ is in } P_1' \text{ where } B\rho^*H'\}$$

and for every  $C \in M'$ :

$$\begin{array}{l} P_2''(C) = \{[a/D] \rightarrow [a/E]wb[b/G]x \mid E, G \in N'; E\tau a; G\tau b; D \rightarrow EwGx \text{ is in } P_2'(C) \text{ where } Ew^{(1)}\rho^*N'\}, \\ P_3''(C) = \{[a/D] \rightarrow [a/E]w \mid E \in N'; E\tau a; D \rightarrow Ew \text{ is in } P_3'(C)\}, \\ P_4''(C) = \{[a/D] \rightarrow wEb[b/F]x \mid a \in \Sigma, F \in N'; F\tau b; D \rightarrow awEFx \text{ is in } P_4'(C) \text{ where } E\rho^*H'\}, \\ P_5''(C) = \{[a/D] \rightarrow w \mid a \in \Sigma; D \rightarrow aw \text{ is in } P_5'(C)\}, \end{array}$$

and define

$$\begin{array}{l} P_1'' = \bigcup_{C \in M'} P_i''(C) \quad \text{for } i \in \{2, 3, 4, 5\}, \\ P_6'' = P_6', \end{array}$$

$$P'' = P_1'' \cup P_2'' \cup P_3'' \cup P_4'' \cup P_5'' \cup P_6'', \\ V'' = V \cup W''; N'' = V'' - \Sigma, \text{ where } W'' = \{[a/A] \mid a \in \Sigma; A \in N'; \text{ there is an } [a/A]\text{-rule in } P''\}.$$

Also define  $H'' = H' \cup \{[a/A] \in W'' \mid A \in H'\}$ .

This transformation is quite similar to that presented by Gray and Harrison [11] for obtaining an operator form grammar. It should be noted that the transformation as shown can operate in general on only  $\Lambda$ -isolated grammars.

The following example illustrates the methods of the transformation. Consider the LR(2) grammar with productions

$$\begin{aligned} \pi_1 &= S \rightarrow Ad \\ \pi_2, \pi_3, \pi_4 &= A \rightarrow aAB \mid b \mid bbc \\ \pi_5 &= B \rightarrow A \end{aligned}$$

The Look-Ahead Reduction Procedure will find that  $\pi_3$  is not LR(1), and thus will specify its left-part as  $H = \{A\}$ . No stratification is required, so  $G' = G$  and  $H' = H$ . Thus  $P_1'$  is found to be

$P_1' = \{\pi_2 = A \rightarrow aAB\}$ , and  $M' = \{B\}$ . (Right-contexts for  $A$ -rules are generated by  $\pi_1$ , which needs no extraction, and by  $\pi_2$ . Right-contexts for  $A$ -rules obtained through  $\pi_2$  must be ultimately extracted through the symbol  $B$ .)

$Q'(B) = \{\pi_5\}$  (In  $\pi_2$ , right-contexts for  $A$ -rules are generated by the symbol  $B$ .  $\pi_5$  is a rule which does not originate terminal right-contexts, but  $\pi_5$  does generate them. Thus extraction must proceed *through*  $\pi_5$ .)

$R'(B) = \{\pi_2, \pi_3, \pi_4\}$  (In  $\pi_2$ , right-contexts for  $A$ -rules are generated by the symbol  $B$ , and since  $B\lambda^*A$ , they are also generated by the symbol  $A$ . The right-contexts which are generated by  $A$  *originate* in the rules  $\pi_2$ ,  $\pi_3$ , and  $\pi_4$ .)

$$\begin{aligned} P_2' &= \emptyset \\ P_3' &= \{\pi_5\} \\ P_4' &= \{\pi_2\} \quad (\pi_2 \text{ is a rule in which the right-contexts for } A\text{-rules are both generated and originated.}) \\ P_5' &= \{\pi_3, \pi_4\} \\ P_6' &= \{\pi_1, \pi_3, \pi_4, \pi_5\}. \end{aligned}$$

Thus, the one-symbol right-contexts for  $A$ -rules originate in  $\pi_2$ ,  $\pi_3$ ,  $\pi_4$  and are  $a$  and  $b$ . The transformation yields:

$$\begin{aligned} P_1'' &= \{A \rightarrow aAa[a/B], A \rightarrow aAb[b/B]\} \quad (\text{ultimate extraction of right-contexts for } A\text{-rules}), \\ P_2'' &= \emptyset, \\ P_3'' &= \{[a/B] \rightarrow [a/A], [b/B] \rightarrow [b/A]\} \quad (\text{propagation of right-contexts through rules of } P_3'), \\ P_4'' &= \{[a/A] \rightarrow Aa[a/B], [a/A] \rightarrow Ab[b/B]\} \quad (\text{origination of the right-context } a \text{ and also ultimate extraction of right-context for } A\text{-rules}), \\ P_5'' &= \{[b/A] \rightarrow \Lambda, [b/A] \rightarrow bc\} \quad (\text{origination of the right-context } b), \\ P_6'' &= \{S \rightarrow Ad, A \rightarrow b, A \rightarrow bbc, B \rightarrow A\} \quad (\text{rules which do not receive ultimate extraction of the right-contexts for } A\text{-rules}). \end{aligned}$$

Note that the rule  $B \rightarrow A$ , although included in  $P''$ , is useless. This poses no problem for these formalizations, but a practical implementation of the Right-Context Extraction Transformation concludes by reducing  $G''$ .

LEMMA 2. Let  $G = (V, \Sigma, P, S)$  ( $N = V - \Sigma$ ) be a reduced  $\Lambda$ -isolated LR(k) grammar, and let  $H \subseteq N$ . Let  $G'' = (V'', \Sigma, P'', S)$  ( $N'' = V'' - \Sigma$ ) and  $H'' \subseteq N''$  be obtained from  $G$  and  $H$  by means of the Right-Context Extraction Transformation. Then

- (a)  $L(G'') = L(G)$ ;
- (b) there is a surjection,  $\phi$ , from  $P_\Lambda''$  onto  $P_\Lambda$  such that if  $\pi \in P_\Lambda''$  is not LR(j), then  $\phi(\pi) \in P_\Lambda$  is not LR(j); and
- (c)  $G''$  is in  $H''$ -right-context-extracted form where for each  $\pi \in P''$ , if  $\phi(\pi) \in P$  is an  $H$ -rule then  $\pi$  is an  $H''$ -rule.

The relationship between  $G'$  of the transformation and  $G$  is given by Lemma 1. We establish some claims relating  $G''$  to  $G'$ .

CLAIM 1. Let  $a \in \Sigma$ ,  $A \in N'$ ,  $v \in \Sigma^*$ .  $A \Rightarrow^{*G'} av$  if and only if  $A \Rightarrow^{*G''} av$ . Moreover, if  $[a/A]$  is in  $N''$ , then  $A \Rightarrow^{*G'} av$  if and only if  $[a/A] \Rightarrow^{*G''} v$ .

PROOF. The proof is by induction on the length of the derivations.

Basis. Suppose  $A \Rightarrow^{G'} av$ . This occurs if and only if  $\pi = A \rightarrow av$  is in  $P'$ . Thus,  $\pi$  is in  $P'_6$ . This occurs if and only if  $A \rightarrow av$  is in  $P_6''$ , which occurs if and only if  $A \Rightarrow^{G''} av$ . Moreover,  $[a/A]$  is in  $N''$  if and only if  $\pi$  is in  $P'_5$  ( $\subseteq P'_6$ ). This occurs if and only if  $[a/A] \rightarrow v$  is in  $P_5''$ , which occurs if and only if  $[a/A] \Rightarrow^{G''} v$ .

Induction step. Suppose that the claim holds for derivations of length  $k$  ( $1 \leq k \leq n$ ) and consider a derivation of length  $n + 1$ . Thus  $A \Rightarrow^{+G'} av$  ( $n + 1$  steps). This may be written

$$A \Rightarrow^{G'} u, \quad u \Rightarrow^{+G'} av \quad (n \text{ steps}). \quad (1)$$

(1) occurs if and only if  $\pi = A \rightarrow u$  is in  $P'$  and  $u$  contains a nonterminal symbol. We have six cases to consider, depending on whether  $\pi$  is in  $P'_1, P'_6, P'_2, P'_3, P'_4$ , or  $P'_5$ .

Case 1.  $\pi$  is in  $P'_1$ . Then there exist  $C \in N'$ ;  $B$  in  $\rho^*H'$  for which  $\pi = A \rightarrow wBCx$ . Also,  $v = v_1v_2bv_3v_4$  ( $b \in \Sigma$ ) where

$$w \Rightarrow^{*G'} av_1, \quad (2)$$

$$B \Rightarrow^{*G'} v_2, \quad (3)$$

$$C \Rightarrow^{*G'} bv_3, \quad (4)$$

$$x \Rightarrow^{*G'} v_4. \quad (5)$$

Such a  $\pi$  is in  $P'_1$  if and only if  $A \rightarrow wBb[b/C]x$  is in  $P_1''$ . By the inductive assumption, (3) and (4) hold if and only if  $B \Rightarrow^{*G''} v_2$  and  $[b/C] \Rightarrow^{*G''} v_3$ . Applying the inductive assumption to the successive nonterminals of  $w$  and  $x$ , we find that (2) and (5) hold if and only if  $w \Rightarrow^{*G''} av_1$  and  $x \Rightarrow^{*G''} v_4$ . Combining these results, we have  $A \Rightarrow^{G''} wBb[b/C]x \Rightarrow^{*G''} av_1v_2bv_3v_4 = av$ . Moreover,  $[a/A]$  is in  $N''$  if and only if  $\pi$  is in  $P'_2 \cup P'_4$  ( $\subseteq P'_1$ ), so one of cases 3 or 5 applies.

Case 2.  $\pi$  is in  $P'_6$ . This occurs if and only if  $\pi = A \rightarrow u$  is in  $P_6''$ . Applying the inductive assumption to the successive nonterminals of  $u$ , we find that (1) holds if and only if  $u \Rightarrow^{*G''} av$ . Thus we have  $A \Rightarrow^{G''} u \Rightarrow^{*G''} av$ . Moreover,  $[a/A]$  is in  $N''$  if and only if  $\pi$  is in  $P'_3 \cup P'_5$  ( $\subseteq P'_6$ ), so one of cases 4 or 6 applies.

Case 3.  $\pi$  is in  $P'_2$ . Then there exist  $B, D \in N'$ ;  $w, x \in V^*$ ;  $Bw^{(1)}$  in  $\rho^*H$  for which  $\pi = A \rightarrow BwDx$ . Also  $v$  may be written  $v = v_1bv_2v_3$  ( $b \in \Sigma$ ) where

$$Bw \Rightarrow^{*G'} av_1, \quad (6)$$

$$D \Rightarrow^{*G'} bv_2, \quad (7)$$

$$x \Rightarrow^{*G'} v_3. \quad (8)$$

Such a  $\pi$  is in  $P'_2$  if and only if  $[a/A] \rightarrow [a/B]wb[b/D]x$  is in  $P_2''$ . As in case 1, we apply the inductive assumption to (6), (7), and (8), and combine to obtain  $[a/A] \Rightarrow^{G''} [a/B]wb[b/D]x \Rightarrow^{*G''} v_1bv_2v_3 = v$ . Moreover,  $P'_2 \subseteq P'_1$ , so case 1 applies and  $A \Rightarrow^{*G''} av$ .

Case 4.  $\pi$  is in  $P'_3$ . Then there exist  $B \in N'$  for which  $\pi = A \rightarrow Bw$ . Also,  $v$  may be written  $v = v_1v_2$  where  $B \Rightarrow^{*G''} av_1$  and  $w \Rightarrow^{*G''} v_2$ . Such a  $\pi$  is in  $P'_3$  if and only if

$[a/A] \rightarrow [a/B]w$  is in  $P_3''$ . Again, as in case 1 we obtain  $[a/A] \Rightarrow^{g''} [a/B]w \Rightarrow^{g''} v_1v_2 = v$ . Moreover,  $P_3' \subseteq P_6'$ , so case 2 applies and  $A \Rightarrow^{g''} av$ .

Case 5.  $\pi$  is in  $P_4'$ . Then there exist  $a \in \Sigma$ ;  $C \in N'$ ;  $B$  in  $\rho^*H'$  for which  $\pi = A \rightarrow awBCx$ . Also,  $v$  may be written  $v = v_1bv_2v_3$  ( $b \in \Sigma$ ) where  $wB \Rightarrow^{g''} v_1$ ,  $C \Rightarrow^{g''} bv_2$ , and  $x \Rightarrow^{g''} v_3$ . Such a  $\pi$  is in  $P_4'$  if and only if  $[a/A] \rightarrow wBb[b/C]x$  is in  $P_4''$ . Again, as in case 1 we obtain  $[a/A] \Rightarrow^{g''} wBb[b/C]x \Rightarrow^{g''} v_1bv_2v_3 = v$ . Moreover,  $P_4' \subseteq P_1'$ , so case 1 applies and  $A \Rightarrow^{g''} av$ .

Case 6.  $\pi$  is in  $P_5'$ . Then  $\pi = A \rightarrow ax$ , where  $x \Rightarrow^{g''} v$ . Such a  $\pi$  is in  $P_5'$  if and only if  $[a/A] \rightarrow x$  is in  $P_5''$ . Again, as in case 1 we obtain  $[a/A] \Rightarrow^{g''} x \Rightarrow^{g''} v$ . Moreover,  $P_5' \subseteq P_6'$ , so case 2 applies and  $A \Rightarrow^{g''} av$ .

These six cases complete the inductive extension, and Claim 1 is proved.  $\square$

CLAIM 2. Every  $x \in \text{CSF}(G'')$  can be uniquely written  $x = x_1x_2 \cdots x_ny$  ( $n \geq 0$ , where if  $n = 0$ ,  $x = y$ ) where  $x_i \in V'^*\{a[a/A] \mid a \in \Sigma; A \in N'; [a/A] \in N''\}$  (for  $i \in \{1, 2, \dots, n\}$ ) and where  $y \in V'^*$ .

PROOF. By inspection of  $P''$  we see that every occurrence of  $[a/A]$  ( $a \in \Sigma$ ;  $A \in N'$ ) in the right-parts of productions is either preceded by an  $a$  or occurs as the leftmost symbol in the right-part. In the latter case, the left-part of the production is of the form  $[a/B]$  ( $B \in N'$ ; same  $a \in \Sigma$ ). Taking this observation into account, the claim is easily proved by induction. We shall leave the details to the reader.  $\square$

Our next result requires the following. Let  $x \in \text{CSF}(G'')$  have the unique decomposition of Claim 2,

$$x = x_1a_1[a_1/A_1]x_2a_2[a_2/A_2] \cdots x_na_n[a_n/A_n]y \quad (x_i \in V'^*; a_i \in \Sigma; A_i \in N'; [a_i/A_i] \in N'' \\ \text{for } i \in \{1, 2, \dots, n\}; y \in V'^*).$$

Define  $\sigma$  as

$$\begin{aligned} \sigma(x) &= \sigma(x_1a_1[a_1/A_1]x_2a_2[a_2/A_2] \cdots x_na_n[a_n/A_n]y) \\ &= x_1A_1x_2A_2 \cdots x_nA_ny. \end{aligned}$$

Clearly for  $x, y \in \text{CSF}(G'')$ , if  $x = y$  then  $\sigma(x) = \sigma(y)$ . We further define  $\phi$ , a surjection from  $P_A''$  onto  $P_A'$ , by

$$\phi(\pi \in P_A'') = \begin{cases} A \rightarrow \sigma(u) & \text{if } \pi = A \rightarrow u \text{ is in } P_1'' \cup P_6'', \\ A \rightarrow \sigma(au) & \text{if } \pi = [a/A] \rightarrow u \text{ is in } P_2'' \cup P_3'' \cup P_4'' \cup P_5'' \text{ } (a \in \Sigma), \\ \Lambda & \text{otherwise.} \end{cases}$$

(The fact that  $\phi$  is a surjection follows easily by construction and the fact that  $G'$  is reduced.)

CLAIM 3. Let  $u \in V''^*$ ;  $y \in \Sigma^*$ ;  $\pi \in P_A''$ . If  $uz \in \text{CSF}(G'')$  has handle  $(\pi, |u|)$  then  $\sigma(uz) \in \text{CSF}(G')$  has handle  $(\phi(\pi), |\sigma(u)|)$ .

PROOF. The proof is by induction on the length of the  $G''$  derivation.

Basis.  $S \in \text{CSF}(G'')$  has handle  $(\Lambda, 1)$  and  $\sigma(S) = S \in \text{CSF}(G')$  has handle  $(\Lambda, 1) = (\phi(\Lambda), |\sigma(S)|)$ .

Induction step. Suppose that the claim holds for  $G''$ -derivations of length  $k$  ( $0 \leq k \leq n$ ) and consider a derivation of length  $n + 1$ . Then  $\pi = X \rightarrow y$  with  $u = xy$  and  $S \Rightarrow^{g''} xXz \Rightarrow^{g''} xyz$  with handle  $(\pi, |xy|)$ . By the inductive assumption,

$$S \Rightarrow^{g''} \sigma(xX)z. \quad (1)$$

We have two cases to consider.

Case 1.  $\pi = A \rightarrow y$  is in  $P_1'' \cup P_6''$ . Then  $\phi(\pi) = A \rightarrow \sigma(y)$  is in  $P_1' \cup P_6'$  and  $\sigma(xA) = \sigma(x)A$ . Thus by (1)  $S \Rightarrow^{g''} \sigma(x)Az \Rightarrow_{\phi(\pi)}^{g''} \sigma(x)\sigma(y)z$ , and  $\sigma(x)\sigma(y)z = \sigma(xyz) \in \text{CSF}(G')$  has handle  $(\phi(\pi), |\sigma(xy)|)$ .

Case 2.  $\pi = [a/A] \rightarrow y$  is in  $P_2'' \cup P_3'' \cup P_4'' \cup P_5''$ . Then  $\phi(\pi) = A \rightarrow \sigma(ay)$  is in  $P_2' \cup P_3' \cup P_4' \cup P_5'$  and, by Claim 2,  $xX = x[a/A] = wa[a/A]$ . Thus,  $\sigma(xX) = \sigma(wa[a/A]) = \sigma(w)A$  and by (1)  $S \Rightarrow^{g''} \sigma(w)Az \Rightarrow_{\phi(\pi)}^{g''} \sigma(w)\sigma(ay)z$  and  $\sigma(w)\sigma(ay)z = \sigma(way)z = \sigma(xyz) \in \text{CSF}(G')$  has handle  $(\phi(\pi), |\sigma(xy)|)$ .  $\square$

PROOF OF LEMMA 2. Claim 1 establishes that  $L(G'') = L(G')$ . By inspection of  $P''$  we see that for  $\pi, \pi' \in P''$ ;  $\pi = X \rightarrow y, \pi' = X' \rightarrow y'$ , if  $\pi \neq \pi'$  and  $\phi(\pi) = \phi(\pi')$ , then  $y^{(|y'|)} \neq y'$  (i.e. by symmetry, neither right-part is a suffix of the other). Suppose that  $\pi \in P_A''$  is not LR( $j$ ). Then there exist  $x, x' \in V''^*$ ;  $z, z' \in \Sigma^*$ ;  $\pi' \in P''$  for which

$$\begin{aligned} &xz \in \text{CSF}(G'') \text{ has handle } (\pi, |x|), \\ &x'z' \in \text{CSF}(G'') \text{ has handle } (\pi', |x'|), \\ &(|x|+j)xz = (|x'|+j)x'z', \text{ and} \\ &(\pi, |x|) \neq (\pi', |x'|). \end{aligned} \quad (1)$$

By Claim 3,  $\sigma(x)z \in \text{CSF}(G')$  has handle  $(\phi(\pi), |\sigma(x)|)$ ,  $\sigma(x')z' \in \text{CSF}(G')$  has handle  $(\phi(\pi'), |\sigma(x')|)$ .

Case 1.  $|x| = |x'|$ . Then by (1)  $x = x'$  whence  $\sigma(x) = \sigma(x')$ . Also  ${}^{(j)}z = {}^{(j)}z'$ . Thus  ${}^{(|\sigma(x)|+j)}\sigma(xz) = {}^{(|\sigma(x')|+j)}\sigma(x'z')$ . Also by (1),  $\pi \neq \pi'$ . Now it cannot be that  $\phi(\pi) = \phi(\pi')$  since then  $x \neq x'$  (by our observation that neither right-part is a suffix of the other). So  $\phi(\pi) \neq \phi(\pi')$ . Consequently  $(\phi(\pi), |\sigma(x)|) \neq (\phi(\pi'), |\sigma(x')|)$  and  $\phi(\pi) \in P_A'$  is not LR( $j$ ).

Case 2.  $|x| < |x'|$ . Then by (1) there exist  $z_1 \in \Sigma^+$ ,  $z_2 \in \Sigma^*$  for which  $z = z_1z_2$  and  $xz_1 = x'$ , whence  $\sigma(x)z_1 = \sigma(x')$ . Since  $|z_1| > 0$  it follows that  $|\sigma(x)| < |\sigma(x')|$ . Thus  $(\phi(\pi), |\sigma(x)|) \neq (\phi(\pi'), |\sigma(x')|)$ . Also by (1) and the fact that  $\sigma$  preserves any terminal suffix of  $x'$ ,  ${}^{(|\sigma(x)|+j)}\sigma(xz) = {}^{(|\sigma(x')|+j)}\sigma(x'z')$ . Consequently  $\phi(\pi) \in P_A'$  is not LR( $j$ ).

Case 3.  $|x| > |x'|$ . As in case 2, we find that  $\phi(\pi) \in P'$  is not LR( $j$ ).

By composition of the surjection obtained here and the one obtained in the Right-Stratification Transformation (Lemma 1), we obtain a new surjection (which we now rename  $\phi$ ) from  $P_A''$  onto  $P_A$ . Also, by the construction of  $P''$  we see that  $G''$  is in  $H''$ -right-context-extracted form where, for  $\pi \in P''$ , if  $\phi(\pi) \in P$  is an  $H$ -rule, then  $\pi$  is an  $H''$ -rule. This completes the proof of Lemma 2.  $\square$

Having obtained a grammar  $G''$  in  $H''$ -right-context-extracted form, we are now ready to develop a transformation to effect the premature scanning of those extracted right-contexts. That is, given some  $H''$ -rule  $(A_n \rightarrow z)$  and a "path" of  $\rho^+H''$ -rules  $(A_0 \rightarrow x_1A_1, \dots, A_{n-1} \rightarrow x_nA_n)$  leading from an extracted right-context  $(a \text{ in } A \rightarrow xA_0ay)$ , we wish to "merge" that right-context (beginning with  $A \rightarrow x[A_0a]y$ ) with the  $\rho^+H''$ -rules (obtaining  $[A_0a] \rightarrow x_1[A_1a], \dots, [A_{n-1}a] \rightarrow x_n[A_na]$  and terminating with  $[A_na] \rightarrow za$ ). However, in order to simplify the proof for that Premature Scanning Transformation, it is convenient to perform a so-called "state-splitting" transformation, isolating the  $\rho^+H''$ -rules by distinguishing their left-parts. This is accomplished by the following.

*Path Isolation Transformation.* Let  $G = (V, \Sigma, P, S)$  ( $N = V - \Sigma$ ) be a reduced  $\Lambda$ -isolated CFG and  $H \subseteq N$ . Apply the Right-Context Extraction Transformation to  $G$  and  $H$  obtaining  $G' = (V', \Sigma, P', S)$  ( $N' = V' - \Sigma$ ) in  $H'$ -right-context-extracted form where  $H' \subseteq N'$  and  $H'$ -rules map onto  $H'$ -rules. Define

$$P_1' = \{A \rightarrow uBav \text{ is in } P' \mid a \in \Sigma; B\rho^*H'\}$$

and let  $M' = \{B \in N' \mid A \rightarrow uBav \text{ is in } P_1' \text{ where } B\rho^*H'\}$ . For each  $B \in M'$  define

$$\begin{aligned} Q'(B) &= \{C \rightarrow wD \text{ in } P' \mid D \in N'; B\rho^*C\}, & P_3'(B) &= Q'(B) - P_2'(B), \\ R'(B) &= \{C \rightarrow wb \text{ in } P' \mid b \in \Sigma; B\rho^*C\}, & P_4'(B) &= R'(B) \cap P_1', \\ P_2'(B) &= Q'(B) \cap P_1', & P_5'(B) &= R'(B) - P_4'(B), \end{aligned}$$

and define

$$P_6' = P' - P_1'.$$

We then define a new grammar,  $G''$ , which differs from  $G'$  in its nonterminal vocabulary and its set of productions.

$$P_1'' = \{A \rightarrow uB'av \mid a \in \Sigma; A \rightarrow uBav \text{ is in } P_1' \text{ where } B\rho^*H'\}$$

and for each  $B \in M'$

$$\begin{aligned} P_2''(B) &= \{C' \rightarrow uE'cwD' \mid D \in N'; c \in \Sigma; C \rightarrow uE'cwD \text{ is in } P_2'(B) \text{ where } E\rho^*H'\}, \\ P_3''(B) &= \{C' \rightarrow wD' \mid D \in N'; C \rightarrow wD \text{ is in } P_3'(B)\}, \\ P_4''(B) &= \{C' \rightarrow uE'cwb \mid E \in N'; c, b \in \Sigma; C \rightarrow uE'cwb \text{ is in } P_4'(B) \text{ where } E\rho^*H'\}, \\ P_5''(B) &= \{C' \rightarrow wb \mid b \in \Sigma; C \rightarrow wb \text{ is in } P_5'(B)\}, \end{aligned}$$

and define

$$\begin{aligned} P_i'' &= \bigcup_{B \in M'} P_i''(B) \quad (\text{for } i \in \{2, 3, 4, 5\}), \\ P_6'' &= P_6', \\ P'' &= P_1'' \cup P_2'' \cup P_3'' \cup P_4'' \cup P_5'' \cup P_6'', \\ V'' &= V' \cup W'' \quad \text{where } W'' = \{A' \mid A \in N' \text{ and there exist an } A' \text{-rule in } P''\}. \end{aligned}$$

Also define

$$H'' = \{A' \in W'' \mid A \in H\}.$$

**LEMMA 3.** *Let  $G = (V, \Sigma, P, S)$  ( $N = V - \Sigma$ ) be a reduced  $\Lambda$ -isolated  $LR(k)$  grammar and let  $H \subseteq N$ . Let  $G'' = (V'', \Sigma, P'', S)$  ( $N'' = V'' - \Sigma$ ) and  $H'' \subseteq N''$  be obtained from  $G$  and  $H$  by means of the Path Isolation Transformation. Then*

- (a)  $L(G'') = L(G)$ ;
- (b) *there is a surjection,  $\phi$ , from  $P_\Lambda''$  onto  $P_\Lambda$  such that if  $\pi \in P_\Lambda''$  is not  $LR(j)$  then  $\phi(\pi) \in P_\Lambda$  is not  $LR(j)$ ; and*
- (c)  *$G''$  is in  $H''$ -right-context-extracted form where for each  $\pi \in P''$ , if  $\phi(\pi) \in P$  is an  $H$ -rule then either  $\pi$  is an  $H''$ -rule or  $\pi$  is  $LR(1)$  with respect to all rules of  $G$ .*

**PROOF.** Path isolation is a particular instance of the "factorization" transformation of Graham [9]. Graham proves that such a transformation preserves both the language and the right-context bound. A slight addition to Graham's proofs yield the needed surjection from  $P_\Lambda''$  onto  $P_\Lambda'$  which, when composed with the surjection from  $P_\Lambda'$  onto  $P_\Lambda$  obtained in right-context extraction (Lemma 2), yields the desired surjection (which we now rename  $\phi$ ) from  $P_\Lambda''$  onto  $P_\Lambda$ . To prove part (c), suppose  $\pi = A \rightarrow v$  in  $P''$  is not an  $H''$ -rule, but that  $\phi(\pi)$  is an  $H$ -rule. Since by construction  $G''$  is in  $H''$ -right-context-extracted form, it follows that  $\pi$  can occur only in derivations like  $S \Rightarrow^{G''} uA \Rightarrow_{\tau}^{G''} w$ . Clearly, under such conditions, if  $\pi$  is not  $LR(1)$ , then  $\pi$ , and by part (b),  $\phi(\pi)$ , are not  $LR(k)$  for any integer  $k$ , which contradicts the hypothesis that  $G$  is  $LR(k)$ .  $\square$

We now present the transformation which is the nucleus of the Look-Ahead Reduction Procedure. Following the transformation, some examples are presented.

**Premature Scanning Transformation.** Let  $G = (V, \Sigma, P, S)$  ( $N = V - \Sigma$ ) be a reduced  $\Lambda$ -isolated CFG and let  $H \subseteq N$ . Apply the Path Isolation Transformation to  $G$  and  $H$ , obtaining  $G' = (V', \Sigma, P', S)$  ( $N' = V' - \Sigma$ ) in  $H'$ -right-context-extracted form, where  $H' \subseteq N'$  with  $H'$ -rules mapping onto  $H$ -rules. Define  $T' = \{A \rightarrow uBav \text{ in } P' \mid a \in \Sigma; B\rho^*H'\}$  and let  $M' = \{Ba \text{ in } N'\Sigma \mid A \rightarrow uBav \text{ is in } T' \text{ where } B\rho^*H'\}$ . For each  $B \in N'$  for which there exists  $a \in \Sigma$  such that  $Ba$  is in  $M'$ , define

$$\begin{aligned} Q'(B) &= \{C \rightarrow wD \text{ in } P' \mid D \in N'; B\rho^*C\}, & P_3'(B) &= Q'(B) - P_2'(B), \\ R'(B) &= \{C \rightarrow wb \text{ in } P' \mid b \in \Sigma; B\rho^*C\}, & P_4'(B) &= R'(B) \cap T', \\ P_2'(B) &= Q'(B) \cap T', & P_5'(B) &= R'(B) - P_4'(B), \end{aligned}$$

and define

$$\begin{aligned} P_i' &= \bigcup_{Ba \in M'} P_i'(B) \quad (\text{for } i \in \{2, 3, 4, 5\}), \\ P_1' &= T' - (P_2' \cup P_4'), \\ P_6' &= P' - (T' \cup P_3' \cup P_4') \end{aligned}$$

(Note that  $P_i' \cap P_j' = \emptyset$  for  $i \neq j$ ;  $i, j \in \{1, 2, 3, 4, 5, 6\}$ ). We then define a new grammar  $G''$  which differs from  $G$  in its nonterminal vocabulary and its set of productions. For each  $Ba \in M'$ , define

$$\begin{aligned} P_2''(Ba) &= \{[Ca] \rightarrow u[Ec]w[Da] \mid c \in \Sigma; D \in N'; C \rightarrow uEcwD \text{ is in } P_2'(B) \text{ where } E\rho^*H'\}, \\ P_3''(Ba) &= \{[Ca] \rightarrow w[Da] \mid D \in N'; C \rightarrow wD \text{ is in } P_3'(B)\}, \\ P_4''(Ba) &= \{[Ca] \rightarrow u[Ec]wa \mid c \in \Sigma; E \in N'; C \rightarrow uEcw \text{ is in } P_4'(B) \text{ where } E\rho^*H'\}, \\ P_5''(Ba) &= \{[Ca] \rightarrow wa \mid C \rightarrow w \text{ is in } P_5'(B)\}, \end{aligned}$$

and define

$$\begin{aligned} P_i'' &= \bigcup_{Ba \in M'} P_i''(Ba) \quad (\text{for } i \in \{2, 3, 4, 5\}), \\ P_1'' &= \{A \rightarrow u[Ba]v \mid a \in \Sigma; A \rightarrow uBav \text{ is in } P_1' \text{ where } B\rho^*H'\}, \\ P_6'' &= P_6', \\ P'' &= P_1'' \cup P_2'' \cup P_3'' \cup P_4'' \cup P_5'' \cup P_6'', \\ V'' &= V \cup W'' \text{ where } W'' = \{[Aa] \mid A \in N'; a \in \Sigma; \text{there exist an } [Aa]\text{-rule in } P''\}. \end{aligned}$$

Before embarking on proofs for the Premature Scanning Transformation, let us illustrate the methods used by the transformation. Consider the LR(2) grammar with productions

$$\begin{aligned} \pi_1, \pi_2 : S &\rightarrow Ab \mid aBbc & \pi_5 : B &\rightarrow C \\ \pi_3, \pi_4 : A &\rightarrow aAbA \mid b & \pi_6, \pi_7 : C &\rightarrow b \mid Cb \end{aligned}$$

The rules  $\pi_4$  and  $\pi_6$  are not LR(1). Thus the Look-Ahead Reduction Procedure will calculate  $H = \{A, C\}$ . Path isolation has no useful effect on this grammar, yielding  $G' = G$  and  $H' = H$ . The Premature Scanning Transformation yields

$$T' = \{\pi_1, \pi_2, \pi_3, \pi_7\} \quad (\text{rules in which right-contexts for } H'\text{-rules originate})$$

$$\text{with } M' = \{Ab, Bb, Cb\},$$

$$Q'(A) = \{\pi_3\} \quad Q'(B) = \{\pi_5\} \quad Q'(C) = \emptyset \quad (\text{rules through which right-contexts for } H'\text{-rules must be propagated}),$$

$$R'(A) = \{\pi_4\} \quad R'(B) = \{\pi_6, \pi_7\} \quad R'(C) = \{\pi_6, \pi_7\} \quad (\text{rules at which propagation of right-contexts will terminate}),$$

$$P_2' = \{\pi_3\} \quad (\text{rules which both originate right-contexts for } H'\text{-rules and also through which such contexts must be propagated}),$$

$$P_3' = \{\pi_5\},$$

$$P_4' = \{\pi_7\} \quad (\text{rules which both originate right-contexts for } H'\text{-rules and also which will terminate propagation of such right-contexts}),$$

$$\begin{aligned} P_5' &= \{\pi_4, \pi_6\}, \\ P_1' &= \{\pi_1, \pi_2\}, \\ P_6' &= \emptyset. \end{aligned}$$

Then

$$\begin{aligned} P_2'' &= P_2''(Ab) = \{[Ab] \rightarrow a[Ab][Ab]\}, & P_5'' &= P_5''(Ab) \cup P_5''(Cb), \\ P_3'' &= P_3''(Bb) = \{[Bb] \rightarrow [Cb]\}, & P_1'' &= \{S \rightarrow [Ab], S \rightarrow a[Bb]c\}, \\ P_4'' &= P_4''(Bb) = P_4''(Cb) = \{[Cb] \rightarrow [Cb]db\}, & P_6'' &= \emptyset. \\ P_5''(Ab) &= \{[Ab] \rightarrow bb\} & P_5''(Cb) &= \{[Cb] \rightarrow bb\}, \end{aligned}$$

The resulting grammar is LR(1).

It is illustrative to note the effect that path isolation has on the Premature Scanning Transformation. It is only as a result of path isolation that we can specify  $P_6' = P'$

–  $(T' \cup P_3' \cup P_6')$ . Consequently the  $P_i'$  are pairwise disjoint. Without path isolation, we would have to specify  $P_6' = P' - P_1'$ , and  $P_6'$  could then overlap  $P_2' \cup P_3' \cup P_4' \cup P_5'$ , making the following proofs much more difficult. As an example, consider the LR(2) grammar with productions

$$\begin{array}{ll} \pi_1, \pi_2 : S \rightarrow Bb \mid D & \pi_5, \pi_6 : A \rightarrow a \mid ab \\ \pi_3, \pi_4 : B \rightarrow A \mid D & \pi_7 : D \rightarrow c \end{array}$$

The rule  $\pi_5$  is not LR(1); thus  $H = \{A\}$ . The Right-Context Extraction Transformation does not alter the grammar, but the Path Isolation Transformation yields

$$\begin{array}{ll} \pi_1, \pi_2 : S \rightarrow B'b \mid D & \pi_8, \pi_9 : B \rightarrow A \mid D \\ \pi_3, \pi_4 : B' \rightarrow A' \mid D' & \pi_{10}, \pi_{11} : A \rightarrow a \mid ab \\ \pi_5, \pi_6 : A' \rightarrow a \mid ab & \pi_{12} : D \rightarrow c \\ \pi_7 : D' \rightarrow c & \end{array}$$

and  $H' = \{A'\}$ . Proceeding with premature scanning,

$$\begin{array}{ll} T' = \{\pi_1\}, & P_4' = \emptyset, \\ M' = \{B'b\}, & P_5' = \{\pi_5, \pi_6, \pi_7\}, \\ P_2' = \emptyset, & P_1' = \{\pi_1\}, \\ P_3' = \{\pi_3, \pi_4\}, & P_6' = \{\pi_2, \pi_8, \pi_9, \pi_{10}, \pi_{11}, \pi_{12}\}, \end{array}$$

and then

$$\begin{array}{ll} P_2'' = \emptyset, & P_6'' = \{[A'b] \rightarrow ab, [A'b] \rightarrow abb, [D'b] \rightarrow cb\}, \\ P_3'' = \{[B'b] \rightarrow [A'b], [B'b] \rightarrow [D'b]\}, & P_1'' = \{S \rightarrow [B'b]\}, \\ P_4'' = \emptyset, & P_6'' = \{S \rightarrow D, B \rightarrow A, B \rightarrow D, A \rightarrow a, A \rightarrow ab, D \rightarrow c\}. \end{array}$$

The resulting grammar is now LR(1).

Now consider the alteration  $P_6' = P' - P_1'$  and apply the altered Premature Scanning Transformation to the original grammar (without applying the Path Isolation Transformation) with  $H' = H = \{A\}$ .

$$\begin{array}{ll} T' = \{\pi_1\} & P_4' = \emptyset, \\ M' = \{Bb\}, & P_5' = \{\pi_5, \pi_6, \pi_7\}, \\ P_2' = \emptyset, & P_1' = \{\pi_1\}, \\ P_3' = \{\pi_3, \pi_4\}, & P_6' = \{\pi_2, \pi_3, \pi_4, \pi_5, \pi_6, \pi_7\} \end{array}$$

(indeed  $P_6' \cap P_3' \neq \emptyset$ ,  $P_6' \cap P_5' \neq \emptyset$ ),

and then

$$\begin{array}{ll} P_2'' = \emptyset, & P_5'' = \{[Ab] \rightarrow ab, [Ab] \rightarrow abb, [Db] \rightarrow cb\}, \\ P_3'' = \{[Bb] \rightarrow [Ab], [Bb] \rightarrow [Db]\}, & P_1'' = \{S \rightarrow [Bb]\}, \\ P_4'' = \emptyset, & P_6'' = \{S \rightarrow D, B \rightarrow A, B \rightarrow D, A \rightarrow a, A \rightarrow ab, D \rightarrow c\}. \end{array}$$

Except for renaming of nonterminals, this grammar is the same as that obtained by performing path isolation followed by premature scanning. Thus, the Path Isolation Transformation is merely a convenience, needed to simplify the following proofs. The simplification lies in noticing that premature scanning causes a nonterminal symbol to be bracketed in some rule of  $P''$  (in either the left-part or the right-part of the rule) if and only if that symbol was primed by the Path Isolation Transformation. Moreover, all instances of primed symbols in rules of  $P'$  become bracketed in the corresponding rules of  $P''$ . Thus we know that for every  $A \in N$ , a rule of  $P''$  can contain some bracketed symbol,  $[Aa]$ , only if no rule of  $P''$  contains the symbol  $A$ , and contrapositively,  $A$  can appear only if no  $[Aa]$  appears.

**LEMMA 4.** Let  $G = (V, \Sigma, P, S)$  ( $N = V - \Sigma$ ) be a reduced  $\Lambda$ -isolated LR( $k$ ) grammar and let  $H \subseteq N$ . Let  $G'' = (V'', \Sigma, P'', S)$  ( $N'' = V'' - \Sigma$ ) be obtained from  $G$  and  $H$  by

means of the Premature Scanning Transformation. Then

- (a)  $L(G'') = L(G)$ ;
- (b) there is a surjection,  $\phi$ , from  $P_A''$  onto  $P_A$  such that if  $\pi \in P_A''$  is not  $LR(j)$  ( $j > 0$ ) then
  - (i)  $\phi(\pi) \in P$  is not  $LR(j+1)$  if  $\pi \in P_2'' \cup P_3'' \cup P_4'' \cup P_5''$ ,
  - (ii)  $\phi(\pi) \in P_A$  is not  $LR(j)$  otherwise; and
  - (c) for each  $\pi \in P''$ , if  $\phi(\pi) \in P$  is an  $H$ -rule, then either  $\pi$  is in  $P_2'' \cup P_3'' \cup P_4'' \cup P_5''$  or  $\pi$  is  $LR(1)$  with respect to all rules of  $G$ .

The relationship between  $G'$  of the transformation and  $G$  is given by Lemma 3. We first prove some claims relating  $G''$  to  $G'$ . Let  $\sigma$  (mapping  $V''$  into  $V \cup V^2$ ) be defined by

$$\sigma(X \in V'') = \begin{cases} Aa & \text{if } X = [Aa] \text{ is in } W'', \\ X & \text{otherwise,} \end{cases}$$

and extend  $\sigma$  to a homomorphism from  $V''^*$  to  $V^*$  by

$$\sigma(xy) = \sigma(x)\sigma(y) \quad \text{for } x, y \in V''^*, \quad \sigma(\Lambda) = \Lambda$$

and let  $\phi$ , a surjection from  $P_A''$  onto  $P_A$ , be defined by

$$\phi(\pi \in P_A'') = \begin{cases} A \rightarrow \sigma(u) & \text{if } \pi = [Aa] \rightarrow ua \text{ is in } P_4'' \cup P_5'' \text{ } (a \in \Sigma), \\ A \rightarrow \sigma(u)B & \text{if } \pi = [Aa] \rightarrow u[Ba] \text{ is in } P_2'' \cup P_3'' \text{ } (a \in \Sigma), \\ A \rightarrow \sigma(u) & \text{if } \pi = A \rightarrow u \text{ is in } P_1'' \cup P_6'', \\ \Lambda & \text{otherwise.} \end{cases}$$

(The fact that  $\phi$  is a surjection follows easily by construction.)

CLAIM 4.  $L(G') \subseteq L(G'')$ .

PROOF. Consider the context-sensitive grammar obtained by deleting the brackets from the rules of  $G''$ . As a result, the equivalent of  $G'$  is obtained.  $\square$

CLAIM 5. Let  $u \in V''^*$ ;  $z \in \Sigma^*$ ;  $\pi \in P''$ . If  $uz \in \text{CSF}(G'')$  has handle  $(\pi, |u|)$  then  $\sigma(uz) \in \text{CSF}(G')$  has handle  $(\phi(\pi), p)$ , where

$$p = \begin{cases} |\sigma(u)| - 1 & \text{if } \pi \in P_2'' \cup P_3'' \cup P_4'' \cup P_5'', \\ |\sigma(u)| & \text{otherwise.} \end{cases}$$

PROOF. The proof is by induction on the length of the  $G''$ -derivation.

Basis.  $S \in \text{CSF}(G'')$  has handle  $(\Lambda, 1)$  and  $\sigma(S) = S \in \text{CSF}(G')$  has handle  $(\Lambda, 1) = (\phi(\Lambda), |\sigma(S)|)$ .

Induction step. Suppose that the claim holds for  $G''$ -derivations of length  $k$  ( $0 \leq k \leq n$ ) and consider a derivation of length  $n+1$ . Then  $\pi = X \rightarrow y$  with  $u = xy$  where  $X \in N''$ ;  $x, y \in V''^*$ ; and  $S \Rightarrow^{*G''} xXz \Rightarrow_{\star} xyz$  with handle  $(\pi, |xy|)$ . By the inductive assumption we have

$$S \Rightarrow^{*G'} \sigma(xXz) = \sigma(xX)z. \quad (1)$$

We have three cases to consider.

Case 1.  $\pi = A \rightarrow y$  is in  $P_1'' \cup P_6''$ . Then  $\phi(\pi) = A \rightarrow \sigma(y)$  is in  $P_1' \cup P_6'$ . Then (1) yields  $S \Rightarrow^{*G'} \sigma(xA)z = \sigma(x)Az \Rightarrow_{\phi(\pi)}^{\sigma(x)\sigma(y)z} \sigma(x)\sigma(y)z = \sigma(xy)z$ . Thus  $\sigma(xy)z \in \text{CSF}(G')$  has handle  $(\phi(\pi), |\sigma(xy)|)$ .

Case 2.  $\pi = [Aa] \rightarrow w[Ba]$  is in  $P_2'' \cup P_3''$  ( $a \in \Sigma$ ). Then  $\phi(\pi) = A \rightarrow \sigma(w)B$  is in  $P_2' \cup P_3'$ . Then (1) yields  $S \Rightarrow^{*G'} \sigma(x[Aa])z = \sigma(x)Aaz \Rightarrow_{\phi(\pi)}^{\sigma(x)\sigma(w)Baz} \sigma(x)\sigma(w)Baz = \sigma(xw)Baz$ . Thus  $\sigma(xy)z = \sigma(xy)z = \sigma(xw[Ba])z = \sigma(xw)Baz \in \text{CSF}(G')$  has handle  $(\phi(\pi), |\sigma(xw)B|) = (\phi(\pi), |\sigma(xw)Ba| - 1) = (\phi(\pi), |\sigma(xy)| - 1)$ .

Case 3.  $\pi = [Aa] \rightarrow wa$  is in  $P_4'' \cup P_5''$  ( $a \in \Sigma$ ). Then  $\phi(\pi) = A \rightarrow \sigma(w)$  is in  $P_4' \cup P_5'$ . Then (1) yields  $S \Rightarrow^{*G'} \sigma(x[Aa])z = \sigma(x)Aaz \Rightarrow_{\phi(\pi)}^{\sigma(x)\sigma(w)az} \sigma(x)\sigma(w)az = \sigma(xw)az$ . Thus  $\sigma(xy)z = \sigma(xy)z = \sigma(xwa)z = \sigma(xw)az \in \text{CSF}(G')$  has handle  $(\phi(\pi), |\sigma(xw)|) = (\phi(\pi), |\sigma(xw)a| - 1) = (\phi(\pi), |\sigma(xy)| - 1)$ .  $\square$

PROOF OF LEMMA 4. By Claim 5 applied to elements of  $L(G'')$  we have  $L(G'') \subseteq L(G')$ , and with Claim 4 this yields  $L(G') = L(G'')$ . To prove part (b) we use the results of Claim 5. Suppose that  $\pi \in P_A''$  is not LR( $j$ ). Then there exist  $x, x' \in V''^*$ ;  $z, z' \in \Sigma^*$ ;  $\pi' \in P''$  for which

$$xz \in \text{CSF}(G'') \text{ has handle } (\pi, |x|), \quad (1)$$

$$x'z' \in \text{CSF}(G'') \text{ has handle } (\pi', |x'|), \quad (2)$$

$$(|x|+j)xz = (|x'|+j)x'z', \quad (3)$$

$$(\pi, |x|) \neq (\pi', |x'|). \quad (4)$$

As in the proof of Lemma 2, we see that for  $\pi, \pi' \in P''$ ;  $\pi = X \rightarrow y$ ,  $\pi' = X' \rightarrow y'$ ; if  $\pi \neq \pi'$  and  $\phi(\pi) = \phi(\pi')$ , then  $y^{(|y'|)} \neq y'$  (i.e. neither right-part is a suffix of the other). The proof now breaks into cases.

Case 1.  $\pi, \pi' \in P_1'' \cup P_6'' \cup \{\Lambda\}$ . By Claim 5,  $\sigma(xz) \in \text{CSF}(G')$  has handle  $(\phi(\pi), |\sigma(x)|)$  and  $\sigma(x'z') \in \text{CSF}(G')$  has handle  $(\phi(\pi'), |\sigma(x')|)$ . Exactly as in the proof of Lemma 2 we find that  $\phi(\pi) \in P_1' \cup P_6' \cup \{\Lambda\}$  is not LR( $j$ ).

Case 2.  $\pi, \pi' \in P_2'' \cup P_3'' \cup P_4'' \cup P_5''$ . By Claim 4 and (1), (2),  $\sigma(xz) = \sigma(x)z \in \text{CSF}(G')$  has handle  $(\phi(\pi), |\sigma(x)|-1)$  and  $\sigma(x'z') = \sigma(x')z' \in \text{CSF}(G')$  has handle  $(\phi(\pi'), |\sigma(x')|-1)$ .

Case 2(a).  $|x| = |x'|$ . Then by (3),  $x = x'$  whence  $\sigma(x) = \sigma(x')$  and  ${}^{(j)}z = {}^{(j)}z'$ . Thus  ${}^{((|\sigma(x)|-1)+1+j)}\sigma(xz) = {}^{((|\sigma(x')|-1)+1+j)}\sigma(x'z')$ . By (4),  $\pi \neq \pi'$ . Now it cannot be that  $\phi(\pi) = \phi(\pi')$  since then  $x \neq x'$  (by our observation that neither right-part is a suffix of the other). So  $\phi(\pi) \neq \phi(\pi')$ . Consequently  $(\phi(\pi), |\sigma(x)|-1) \neq (\phi(\pi'), |\sigma(x')|-1)$  and  $\phi(\pi) \in P_2' \cup P_3' \cup P_4' \cup P_5'$  is not LR( $j+1$ ).

Case 2(b).  $|x| \neq |x'|$ . Just as case 2(a) was handled very much like case 1 in the proof of Lemma 2, so too, Case 2(b) is handled much like cases 2 and 3 in the proof of Lemma 2, and we find that  $\phi(\pi) \in P_2' \cup P_3' \cup P_4' \cup P_5'$  is not LR( $j+1$ ).

Case 3.  $\pi \in P_1'' \cup P_6'' \cup \{\Lambda\}$ ;  $\pi' \in P_2'' \cup P_3'' \cup P_4'' \cup P_5''$ . By Claim 5 and (1) and (2),  $\sigma(xz) = \sigma(x)z \in \text{CSF}(G')$  has handle  $(\phi(\pi), |\sigma(x)|)$  and  $\sigma(x'z') = \sigma(x')z' \in \text{CSF}(G')$  has handle  $(\phi(\pi'), |\sigma(x')|-1)$ . By (3) and the fact that  $\sigma$  preserves any terminal suffix of  $x'$ , we have  ${}^{(|\sigma(x)|+j)}\sigma(xz) = {}^{(|\sigma(x')|-1)+j)}\sigma(x'z')$ . Now it must be that  $\phi(\pi) \neq \phi(\pi')$ , since  $P_1'' \cup P_6''$ -rules have unbracketed left-parts, whereas  $P_2'' \cup P_3'' \cup P_4'' \cup P_5''$ -rules have bracketed left-parts. If it were that  $\phi(\pi) = \phi(\pi')$  then  $\pi$  would be of the form  $A \rightarrow y$  and  $\pi'$  of the form  $[Aa] \rightarrow y'$ ; but as a result of Path Isolation, nonterminal symbols cannot be both bracketed and unbracketed in  $N''$ . Thus  $\phi(\pi) \neq \phi(\pi')$  and  $(\phi(\pi), |\sigma(x)|) \neq (\phi(\pi'), |\sigma(x')|-1)$  and  $\phi(\pi) \in P_1' \cup P_6' \cup \{\Lambda\}$  is not LR( $j$ ).

Case 4.  $\pi \in P_2'' \cup P_3'' \cup P_4'' \cup P_5''$ ;  $\pi' \in P_1'' \cup P_6'' \cup \{\Lambda\}$ . By Claim 5 and (1) and (2),  $\sigma(xz) = \sigma(x)z \in \text{CSF}(G')$  has handle  $(\phi(\pi), |\sigma(x)|-1)$  and  $\sigma(x'z') = \sigma(x')z' \in \text{CSF}(G')$  has handle  $(\phi(\pi'), |\sigma(x')|)$ . By (3) and the fact that  $\sigma$  preserves any terminal suffix of  $x'$ , we have  ${}^{((|\sigma(x)|-1)+1+j)}\sigma(xz) = {}^{((|\sigma(x')|-1)+1+j)}\sigma(x'z')$ . As in case 3,  $\phi(\pi) \neq \phi(\pi')$  whence  $(\phi(\pi), |\sigma(x)|-1) \neq (\phi(\pi'), |\sigma(x')|)$  and  $\phi(\pi) \in P_2' \cup P_3' \cup P_4' \cup P_5'$  is not LR( $j+1$ ).

Part (c) of Lemma 4 follows by construction, part (c) of Lemma 3, and part (b) of Lemma 4. This completes the proof of Lemma 4.  $\square$

Although we are now ready to describe an LR( $k$ ) to LR( $k-1$ ) algorithm for reduced,  $\Lambda$ -isolated grammars, we cannot yet describe the LR( $k$ ) to LR(1) algorithm. While such an algorithm is simply an iterative version of LR( $k$ ) to LR( $k-1$ ), it must also account for the possibility that, between iterations, the Right-Context Extraction Transformation may introduce new  $\Lambda$ -rules, destroying the  $\Lambda$ -isolation property. Thus, it is necessary to perform yet another transformation, removing  $\Lambda$ -rules, to recover a  $\Lambda$ -isolated grammar. The standard  $\Lambda$ -elimination algorithm for CFGs (cf. Hopcroft and Ullman [14, pp. 62, 63]) may be used. The transformation (which we call a " $\Lambda$ -Isolation

Transformation") preserves both the language (modulo  $\Lambda$ ) and the right-context bound (cf. Graham [10, Cor. 4.4]).

Finally, we can present the procedure for reducing the right-context bound of an  $LR(k)$  grammar. We have parameterized the algorithm by specifying that it should halt after having obtained an  $LR(p)$  grammar.

*Look-Ahead Reduction Procedure.* Given an  $LR(k)$  grammar,  $G = (V, \Sigma, P, S)$ ,

**for**  $i := k - 1$  **down to**  $p$

- (1) Apply the  $\Lambda$ -Isolation Transformation to  $G$ , obtaining  $G' = (V', \Sigma, P', S')$  ( $N' = V' - \Sigma$ ).
- (2) Compute  $H' = \{A \in N' \mid \text{some } A\text{-rule of } P' \text{ is not } LR(i)\}$ .
- (3) Apply the Premature Scanning Transformation to  $G'$  and  $H'$ , obtaining a new grammar,  $G = (V, \Sigma, P, S)$ .

**halt**

**THEOREM 1.** *Let  $G = (V, \Sigma, P, S)$  be a CFG. If  $G$  is  $LR(k)$  for some  $k$ , then the Look-Ahead Reduction Procedure halts for  $p \geq 1$ , yielding an  $LR(p)$  grammar  $G''$  for which  $L(G'') = L(G)$ .*

**PROOF.** The fact that  $L(G'') = L(G)$  is clear from Lemma 4. Thus, to prove the theorem, it is sufficient to show that the  $(k-j)$ -th iteration of the Look-Ahead Reduction Algorithm effectively converts an  $LR(j+1)$  grammar,  $G$ , to an  $LR(j)$  grammar,  $G''$ . The  $(k-j)$ -th iteration certainly halts since the transformations each involve only finite computations on finite grammars. Moreover, the  $\Lambda$ -Isolation Transformation preserves the right-context bound. Suppose that after applying the Premature Scanning Transformation,  $G''$  is not  $LR(j)$  ( $j \geq 1$ ). Then there is some rule,  $\pi \in P_\Lambda''$ , which is not  $LR(j)$ . Consider the partitioning of  $P''$ ,  $(P_1'', P_2'', P_3'', P_4'', P_5'', P_6'')$  accomplished by premature scanning. It cannot be that  $\pi = \Lambda$  since then (by Lemma 4),  $\phi(\pi) = \Lambda$  is not  $LR(j)$  ( $j \geq 1$ ) whence  $G'$  and  $G$  are not  $LR(k)$  for any  $k$ , contradicting the hypothesis that  $G$  is  $LR(k)$ .

It cannot be that  $\pi \in P_2'' \cup P_3'' \cup P_4'' \cup P_5''$  since then (by Lemma 4)  $\phi(\pi) \in P'$  is not  $LR(j+1)$ , contradicting the hypothesis that  $G$  and  $G'$  are  $LR(j+1)$ . Thus,  $\pi \in P_1'' \cup P_6''$ . But then (by Lemma 4),  $\phi(\pi) \in P'$  is not  $LR(j)$ , and by step 2 of the Look-Ahead Reduction Procedure,  $\phi(\pi)$  is an  $H'$ -rule. Then (by Lemma 4), either  $\pi$  is  $LR(1)$  with respect to all rules of  $G''$  or  $\pi \in P_2'' \cup P_3'' \cup P_4'' \cup P_5''$ , each of which again leads to a contradiction. Thus it must be that after the  $(k-j)$ -th iteration, each rule  $\pi$  of  $P''$  is  $LR(j)$  with respect to all rules of  $G''$ .  $\square$

It is natural to ask under what conditions the Look-Ahead Reduction Procedure will halt for  $p = 0$ , yielding an  $LR(0)$  grammar. Clearly this would follow if part (c) of Lemma 4 were to read:

- (c) *for each  $\pi \in P''$ , if  $\phi(\pi) \in P$  is an  $H$ -rule, then either  $\pi$  is in  $P_2'' \cup P_3'' \cup P_4'' \cup P_5''$  or  $\pi$  is  $LR(0)$  with respect to all rules of  $G$ .*

This in turn would follow if part (c) of Lemma 3 were to read:

- (c)  *$G''$  is in  $H''$ -right-context-extracted form where for each  $\pi \in P''$ , if  $\phi(\pi) \in P$  is an  $H$ -rule, then either  $\pi$  is an  $H''$ -rule or  $\pi$  is  $LR(0)$  with respect to all rules of  $G$ .*

There is a quite simple language constraint which is sufficient to yield these strengthened conclusions.

**THEOREM 2.** *Let  $G = (V, \Sigma, P, S)$  be a CFG. If  $G$  is  $LR(k)$  for some  $k$  and  $L(G)$  is prefix-free,<sup>8</sup> then the Look-Ahead Reduction Procedure halts for  $p \geq 0$ , yielding an  $LR(p)$  grammar,  $G''$ , for which  $L(G'') = L(G)$ .*

**PROOF.** In view of the proofs of Lemma 4 and Theorem 1, it is necessary to show that the prefix-free condition on  $L(G)$  is sufficient to yield the cited modification to Lemma 3.

<sup>8</sup> A language,  $L \subseteq \Sigma^*$ , is said to be *prefix-free* [13] if and only if  $x \in L$  and  $xy \in L$  implies  $y = \Lambda$ .

Suppose (within the hypotheses of Lemma 3 and the prefix-free hypothesis) that  $\pi = A \rightarrow v$  in  $P''$  is not an  $H''$ -rule, but that  $\phi(\pi)$  is an  $H$ -rule. As in the proof of Lemma 3, since by construction  $G''$  is in  $H''$ -right-context-extracted form, it follows that  $\pi$  can occur only in derivations like

$$S \Rightarrow^{*G''} uA \Rightarrow^{G''} uw \text{ with handle } (\pi, |uw|). \quad (1)$$

Suppose that  $\pi$  is not LR(0). Then there exist  $w \in \Sigma^*$  and  $\pi' \in P''$  for which

$$uwv \in \text{CSF}(G'') \text{ has handle } (\pi', j) \quad (2)$$

and

$$(\pi, |uw|) \neq (\pi', j). \quad (3)$$

Since  $G''$  is reduced, it follows that there exists  $y \in \Sigma^*$  for which  $uw \Rightarrow^{*G''} y$ . Then by (1) and (2),  $y \in L(G'')$  and  $yw \in L(G'')$ . Since  $L(G) = L(G'')$  is prefix-free it follows that  $w = \Lambda$ . But then, by (1), (2), and (3), we find that  $\pi$  is not LR( $k$ ) for any  $k$ . Thus, by part (b) of Lemma 3, we are led to the contradiction that  $\phi(\pi)$  is not LR( $k$ ) for any  $k$ .  $\square$

#### 4. Concluding Remarks

We have presented a number of the preceding transformations separately for the sake of proving them correct. However, in practice, some economization is possible. We have already remarked that the Path Isolation Transformation followed by the Premature Scanning Transformation can be combined into a single modification of the Premature Scanning Transformation in which we define  $P'_6 = P' - P'_1$  instead of  $P'_6 = P' - (T' \cup P'_3 \cup P'_5)$ . However, it is more difficult to prove that the modified version works correctly. Another simplification is possible in the Look-Ahead Reduction Procedure. The modified procedure is:

Given an LR( $k$ ) grammar,  $G$ ,

**while**  $G$  is not LR( $p$ )

- (1) Apply the  $\Lambda$ -Isolation Transformation to  $G$ , yielding  $G' = (V', \Sigma, P', S')$  ( $N' = V' - \Sigma$ ).
- (2) Compute  $H'$  as some (arbitrary, but nonempty) subset of  $\{A \in N' \mid \text{some } A\text{-rule of } P' \text{ is not LR}(p)\}$ .
- (3) Apply the Premature Scanning Transformation to  $G'$  and  $H'$ , yielding a new grammar,  $G = (V, \Sigma, P, S)$

**halt**

It is a very difficult task to prove that this modified Look-Ahead Reduction Procedure halts (and we have not tried to do so). However, we think that this version is computationally more efficient than the original algorithm (if for no other reason than the simplified need to detect only non-LR( $p$ ) ( $p = 0$  or  $1$ ) instead of non-LR( $i$ ) rules). In our implementation [19–22, 24] we stop at the first encountered non-LR( $p$ ) rule,  $A \rightarrow u$ , and set  $H = \{A\}$ .

In our preliminary remarks, we noted that it is not necessary to apply premature scanning to *all* extracted right-contexts, but only to those which are involved in non-LR( $p$ ) conflicts. The modification that is needed in the Premature Scanning Transformation is to change the definition of the set  $T'$  to

$$T' = \{A \rightarrow uBav \mid a \in \Sigma; Bp^*H'; \text{ and } a \text{ is a conflicting right-context for some } H'\text{-rule}\}.$$

Theorem 2 asserts that if  $L(G)$  is prefix-free (and  $G$  is LR( $k$ )) then the Look-Ahead Reduction Procedure is capable of producing an equivalent LR(0) grammar. We remind the reader that any language can be made prefix-free by the concatenation of an end-

marker to each sentence of the language. Formally, let  $G = (V, \Sigma, P, S)$  be a CFG and let  $S'$  and  $\$$  be symbols not in  $V$ . Then for  $G' = (V \cup \{S', \$\}, \Sigma \cup \{\$, \$\}, P \cup \{S' \rightarrow S\$, S'\}, L(G'))$  is prefix-free. Geller and Harrison [7] call such a grammar,  $G'$ , the “ $\$$ -augmented grammar of  $G$ .” In practice, virtually all programming languages have such endmarkers (e.g. “end-of-record mark”).

An interesting side effect of the Look-Ahead Reduction Procedure is that it can be modified to yield a grammar in DeRemer’s “simple” LR(1) (SLR(1)) form [3, 4]. The modified procedure is:

Given an LR( $k$ ) grammar,  $G$ ,

**while**  $G$  is not SLR(1)

- (1) Apply the  $\Lambda$ -Isolation Transformation to  $G$ , yielding  $G' = (V', \Sigma, P', S')$  ( $N' = V' - \Sigma$ ).
- (2) Compute  $H'$  as some (arbitrary, but nonempty) subset of  $\{A \in N' \mid \text{some } A\text{-rule of } P' \text{ is not SLR}(1)\}$ .
- (3) Apply the Premature Scanning Transformation to  $G'$  and  $H'$ , yielding a new grammar,  $G = (V, \Sigma, P, S)$ .

**halt**

Although this procedure will often halt for an arbitrary LR( $k$ ) grammar,  $G$ , halting is guaranteed only if  $L(G)$  is prefix-free. That is, in some cases, an LR(0), hence SLR(0) (see [3, 4]), hence SLR(1) grammar is the best that can be obtained. As an example of conversion to SLR(1), consider the LR(1) grammar with the following productions. This grammar is not SLR( $k$ ) for any integer,  $k$ .

$$\begin{aligned} \pi_1, \pi_2, \pi_3: \quad & S \rightarrow aAa \mid aBb \mid bBa \\ \pi_4, \pi_5: \quad & A \rightarrow c \mid cA \\ \pi_6, \pi_7: \quad & B \rightarrow c \mid cB \end{aligned}$$

Rules  $\pi_4$  and  $\pi_6$  are not SLR(1). Suppose that we compute  $H = \{B\}$ . No right-context extraction is required. After premature scanning of the conflicting right-contexts (merely the rightmost  $a$  of  $\pi_3$ ), we obtain the grammar with productions

$$\begin{aligned} \pi_1', \pi_2', \pi_3': \quad & S \rightarrow aAa \mid aBb \mid b[Ba], & \pi_6', \pi_7': \quad & B \rightarrow c \mid cB \\ \pi_4', \pi_5': \quad & A \rightarrow c \mid cA & \pi_8', \pi_9': \quad & [Ba] \rightarrow ca \mid c[Ba] \end{aligned}$$

which is SLR(1).

In our implementation, we require a (1, 1) bounded right-context (BRC) grammar, which is obtained from an LR( $k$ ) grammar by (1) transforming to SLR(1) via the above algorithm, and then (2) transforming to (1, 1) BRC by a scheme that is much like the one presented by Graham [9]. Consider once again the grammar with productions  $\pi_1, \dots, \pi_7$  above. Using Graham’s original scheme [9], “state splitting” is used to obtain the SLR(1) grammar with productions

$$\begin{aligned} S &\rightarrow aAa \mid aBb \mid bB'a & B &\rightarrow c \mid cB \\ A &\rightarrow c \mid cA & B' &\rightarrow c \mid cB' \end{aligned}$$

and subsequently, one obtains the (1, 1) BRC grammar with productions

$$\begin{aligned} S &\rightarrow aAa \mid aBb \mid bB'a & B' &\rightarrow c \mid CB' \\ A &\rightarrow c \mid cA & C &\rightarrow c \\ B &\rightarrow c \mid cB \end{aligned}$$

In her more recent work [10], Graham shows how to *directly* obtain (1, 1) BRC from LR(1), but to obtain a grammar of reasonable size requires considerable optimization of the transformation.

With our implementation we first obtain the SLR(1) grammar with productions

$\pi_1', \dots, \pi_9'$  above, and subsequently the (1, 1) BRC grammar with productions

$$\begin{array}{ll} S \rightarrow aAa \mid aBb \mid b[Ba] & [Ba] \rightarrow ca \mid C[Ba] \\ A \rightarrow c \mid cA & C \rightarrow c \\ B \rightarrow c \mid cB \end{array}$$

Our final observation concerns the relation of these transformations to the notion of grammatical covers (originally due to Reynolds and Haskell [23]). Gray and Harrison [11] present a definition of cover which is similar to the following.

Let  $G = (V, \Sigma, P, S)$  and  $G'' = (V'', \Sigma, P'', S)$  be CFGs and let  $\psi$  be a mapping from  $P''$  to  $P_A$ . Extend  $\psi$  to a homomorphism from  $P''^*$  to  $P^*$  by requiring  $\psi(\Lambda) = \Lambda$  and for  $x, y \in P''$ ,  $\psi(xy) = \psi(x)\psi(y)$ .  $G''$  is said to *completely cover*  $G$  under  $\psi$  if and only if

(a)  $L(G'') = L(G)$ ; and

(b) for each  $x \in L(G)$ , (i) if  $S \Rightarrow_{(\pi, \pi_1, \dots, \pi_n)}^a x$  then there exist  $(\pi_i'')^m_{i=1}$  in  $P''^m$  ( $m \geq n$ ) for which  $S \Rightarrow_{(\pi_i'')^m_{i=1}}^a x$  and  $\psi((\pi_i'')^m_{i=1}) = (\pi_i)_{i=1}^m$ , (ii) if  $S \Rightarrow_{(\pi, \pi_1, \dots, \pi_n)}^{a''} x$  then  $S \Rightarrow_{\psi((\pi, \pi_1, \dots, \pi_n))}^{a''} x$ .

Gray and Harrison actually present a more general notion within which a  $G''$ -derivation may induce (via  $\psi$ ) only a portion of the corresponding  $G$ -derivation. In their terminology, "sparse"  $G$ -derivations are covered.

We note that, with the exception of  $\Lambda$ -Isolation, each transformation presented here produces as output a grammar  $G''$  which completely covers the input grammar,  $G$ . In each case, a simple modification of the associated surjection,  $\phi$ , provides the cover mapping,  $\psi$  from  $P''$  to  $P_A$ . For the  $\Lambda$ -Isolation Transformation, a sparse covering is accomplished. The only portions of input grammar derivations which are not covered are subderivations which lead to  $\Lambda$ . We have found that, in practice, it is quite easy to relocate any semantic actions that may have been associated with such neglected  $\Lambda$ -subderivations.

## REFERENCES

(Note. References [2, 8, 15] are not cited in the text.)

1. AHO, A V, DENNING, P J., AND ULLMAN, J.D. Weak and mixed strategy precedence parsing. *J. ACM* 19, 2 (April 1972), 225-243.
2. AHO, A.V., AND ULLMAN, J.D. *The Theory of Parsing, Translation, and Compiling, Vol. 2*. Prentice-Hall, Englewood Cliffs, N.J., 1973.
3. DEREMER, F.L. Practical translators for LR( $k$ ) languages. Doctoral Diss., MIT, Cambridge, Mass., Sept. 1969.
4. DEREMER, F.L. Simple LR( $k$ ) grammars. *Comm. ACM* 14, 7 (July 1971), 453-460.
5. FLOYD, R.W. Syntactic analysis and operator precedence. *J. ACM* 10, 3 (July 1963), 316-333.
6. FLOYD, R.W. Bounded-context syntactic analysis. *Comm. ACM* 7, 2 (Feb 1964), 62-67.
7. GELLER, M.M., AND HARRISON, M.A. Characterizations of LR(0) languages (extended abstract). Proc. of the 14th Ann. Symp. on Switching and Automata Theory, Iowa City, Ia., Oct. 1973, pp. 103-108 (sponsored by IEEE, New York).
8. GINSBURG, S., AND GREIBACH, S.A. Deterministic context-free languages. *Inform. and Contr.* 9 (1966), 620-648.
9. GRAHAM, S.L. Precedence languages and bounded right context languages. Doctoral Diss., Stanford U., Stanford, Calif., July 1971.
10. GRAHAM, S.L. On bounded right context languages and grammars. *SIAM J. Comput.* 3 (1974), 224-254.
11. GRAY, J.N., AND HARRISON, M.A. On the covering and reduction problems for context-free grammars. *J. ACM* 19, 4 (Oct. 1972), 675-698.
12. HARRISON, M.A. On the parsing of deterministic languages. Invited presentation, ACM Comput. Sci. Conf., Columbus, Ohio, 1973 (oral presentation).
13. HARRISON, M.A., AND HAVEL, I.M. Strict deterministic grammars. *J. Comput. and Syst. Scis.* 7 (1973), 237-277.
14. HOPCROFT, J.E., AND ULLMAN, J.D. *Formal Languages and Their Relation to Automata*. Addison-Wesley, Reading, Mass., 1969.

15. ICHBIAH, J.D., AND MORSE, S.P. A technique for generating almost optimal Floyd-Evans productions for precedence grammars. *Comm. ACM* 13, 8 (Aug. 1970), 501-508.
16. KNUTH, D.E. On the translation of languages from left to right. *Inform. and Contr.* 8 (1965), 607-639.
17. LALONDE, W.R. An efficient LALR parser generator. Tech. Rep. CSRG-2, U. of Toronto, Toronto, Ont., Canada, 1971.
18. MCAFEE, J., AND PRESSER, L. An algorithm for the design of simple precedence grammars. *J. ACM* 19, 3 (July 1972), 385-395.
19. MICKUNAS, M.D. User's manual for the PUCSD parser generating system. Tech. Rep., Purdue U., West Lafayette, Ind., Aug. 1973.
20. MICKUNAS, M.D. Techniques for compressing bounded-context acceptors. Doctoral Diss., Purdue U., West Lafayette, Ind., May 1973.
21. MICKUNAS, M.D., AND SCHNEIDER, V.B. On the ability to cover  $LR(k)$  grammars with  $LR(1)$ ,  $SLR(1)$ , and  $(1, 1)$  bounded-context grammars. Proc. of the 14th Ann. Symp. on Switching and Automata Theory, Iowa City, Ia., Oct. 1973, pp. 109-121 (sponsored by IEEE, New York).
22. MICKUNAS, M.D., AND SCHNEIDER, V.B. A parser-generating system for constructing compressed compilers. *Comm. ACM* 16, 11 (Nov. 1973), 669-676.
23. REYNOLDS, J.C., AND HASKELL, R. Grammatical coverings. Unpublished manuscript, 1970.
24. SCHNEIDER, V.B. A system for designing fast programming language translators. Proc. AFIPS 1969 SJCC, Vol. 34, AFIPS Press, Montvale, N.J., pp. 777-792.
25. WIRTH, N., AND WEBER, H. EULER: a generalization of ALGOL and its formal definition: Parts I, II. *Comm. ACM* 9, 1, 2 (Jan., Feb. 1966), 13-23, 25, 89-99.

RECEIVED MAY 1973; REVISED NOVEMBER 1975