Symmetrical precedence relations on general phrase structure grammars

R. Haskell

Department of Computing and Control, Imperial College of Science and Technology, 48 Princes Gardens, London, SW7 1LU

In this paper precedence relations are defined on *general phrase structure grammars*. Unlike the formulations used for context free grammars four precedence relations are used and these are symmetrically defined. A two stack parsing method for simple precedence phrase structure languages is presented. An algorithm to transform any phrase structure grammar to precedence form is also described.

(Received April 1973)

1. Introduction

The precedence (or hierarchy) of arithmetic operators is a familiar notion and predates the theory of formal grammars and languages. This notion was first incorporated into the theory of context free grammars by Floyd (1963) in his first paper on operator precedence. Floyd used the operator grammars and defined precedence relations on the terminal symbols of the grammars. He used these precedence relations to define an elegant parsing algorithm for the operator precedence languages. The theory for context free grammars was further developed by Wirth and Weber (1966) and later by Colmerauer (1970). Unlike Floyd, they defined precedence relations on terminal and non-terminal symbols of the grammar. Parsing algorithms using these precedence relations were also described by them.

The previous definitions of precedence relations were all based on context free grammars. Furthermore, most theoretical studies into formal definitional methods and parsing have in the past concentrated on context free grammars.

It is well known that the syntax of most high level programming languages cannot be fully described by using context free grammars. General phrase structure grammars on the other hand are adequately powerful for describing programming language syntax. For this reason a study of phrase structure grammars is made in this paper as it is felt that an advance in this theory would be useful in language definition.

Precedence relations are defined on a general phrase structure grammar. Unlike formulations used for context free grammars four symmetrically defined precedence relations are used. A two stack parser which uses these precedence relations is then presented. The precedence relations and parser are not dissimilar to those used by Colmerauer (1970) for context free grammars.

An algorithm for converting any phrase structure grammar into precedence form is also presented. The algorithm uses '(context) expansions' to eliminate precedence clashes. It is not dissimilar to one used by Lim (1972) for context free grammars.

2. Notation and terminology

In this section we describe the notation, terminology and conventions that will be used in the rest of the paper. Our objective here is to provide a notation which is convenient for describing phrase structure grammars *and* the related parser. For this reason the notation is a variant on what is traditionally used, suitable generalisation and constraints being introduced where appropriate.

The subsections following are (2.0)-(Phrase structure) grammars, (2.1)-Conventions, (2.2)-The relations λ, ρ; left and right sets and (2.3)-(Symmetrical) precedence relations and precedence grammars.

2.0. (*Phrase structure*) *grammars*

A (phrase structure) *grammar* $G = (V, T, S, P, \vdash, \dashv)$ consists of:

1. A finite set V known as the vocabulary
2. A set of terminals $T \subseteq V$.
3. $S \subseteq \vdash V^* \dashv$ is a finite set of initial (or starting) strings.
4. A set of productions P. Each production must have the form $p \to q$ where $p \neq q$; p, q are not the null string, and one of (*a*) to (*d*) hold.
   (*a*) $p, q \in V^*$
   (*b*) $p, q \in \vdash V^*$
   (*c*) $p, q \in V^* \dashv$
   (*d*) $p, q \in \vdash V^* \dashv$
5. A *left endmarker* $\vdash$ and a *right endmarker* $\dashv$. We require $\vdash, \dashv \notin V$.

Let $v, v' \in \vdash V^* \dashv$. The we write $v \to v'$ providing there exists a production $p \to q$ in P, some $w \in \{e\} \cup \vdash V^*$ and some $w' \in \{e\} \cup V^* \dashv$ such that $v = wpw'$ and $v' = wqw'$.

If $v \to v_0 \to v_1 \ldots \to v_n = v'$, where $n \geqslant 1$ and $v_i \in \vdash V^* \dashv$ for $0 \leqslant i \leqslant n$ we write $v \to {}^+v$. Furthermore if $v = v'$ or $v \to {}^+v$: we write $v \to {}^*v'$ and say v' is derived from or equals v.

A *sentential form* is any element $v \in \vdash V^* \dashv$ such that $s \to {}^*v$ for some $s \in S$.

The *language* of G is

$$L(G) = \{t \in T^* \colon \text{for some } s \in S,\ s \to {}^*\vdash t \dashv\}\,.$$

A phrase structure grammar is said to be *simple* if $p \to q$, $p' \to q$ are not both in P when $p \neq p'$.

A phrase structure grammar is said to be *loop free* if $v \to {}^+v$ never occurs for $v \in \vdash V^* \dashv$.

A phrase structure grammar is said to be *context sensitive* if the length of p is less than or equal to the length of q for each production $p \to q$ in P.

A phrase structure grammar is said to be *context free* if productions only have the form $A \to q$ where $A \in V \backslash T$ and $q \in V^*$.

The distinctive points to note in the preceding definitions are:

(*a*) The use of terminal symbols on both sides of a production and in elements of S.
(*b*) The constraint that p, q must not be the null string.
(*c*) The use of endmarkers $\vdash, \dashv$. Endmarkers are introduced firstly because it simplifies the description of the parser and secondly because it enables us to restrict a substitution to either end of a string if required.
(*d*) The use of a finite set $S \subseteq \vdash V^* \dashv$ of initial (or starting) strings.
(*e*) The slightly different definitions of sentential forms and language of G and the way endmarkers and the relation $\to^*$ feature in their definition.
(*f*) The use of the relation $\to^+$ and not $\to^*$ in the definition of a loop free phrase structure grammar.

Downloaded from https://academic.oup.com/comjnl/article/17/3/234/389303 by guest on 21 August 2022

2.1. *Conventions*
Henceforth phrase structure grammars shall be referred to as grammars. Unless otherwise stated G (with possible subscripts, superscripts, etc.) shall denote a grammar and V, T, S, P (with corresponding subscripts, superscripts, etc.) will denote the vocabulary, the terminal set, the initial (or starting) string set and the production set respectively. The left and right endmarkers will *always* be denoted by $\vdash$ and $\dashv$ respectively. Other capital letters will be used to denote elements of $V \cup \{\vdash, \dashv\}$.
The letter t will be used to denote an element T^*, the letter e will be used to denote the null string and the letter s will be used to denote an element of S. Other small letters will be used to denote elements of $\{\vdash, e\}V^*\{e, \dashv\}$, i.e. strings of V^* possibly delimited with endmarkers.

2.2. *The relations* λ, ρ; *left and right sets*
Let G be a grammar and $A, B \in V \cup \{\vdash, \dashv\}$. We define:

$A\lambda B$ iff $Ax \rightarrow By$ is in P for some x, y.
$A\rho B$ iff $xA \rightarrow yB$ is in P for some x, y.

The Left and Right Sets are respectively defined by:

$l(A) = \{B_n: A\lambda B_1\, \lambda B_2 \ldots \lambda B_n$ for some $n \geqslant 1\}$.
$r(A) = \{B_n: A\rho B_1\, \rho B_2 \ldots \rho B_n$ for some $n \geqslant 1\}$.

The distinctive points to note here are:

1. $l(A)$, $r(A)$ are defined for *all* elements in $V \cup \{\vdash, \dashv\}$ *including* the terminal symbols.
2. If $A \in l(B)$ and $B \in l(C)$ then $A \in l(C)$.
   If $A \in r(B)$ and $B \in r(C)$ then $A \in r(C)$.

The reader familiar with the algebra of relations will realise that $l(A) = \{B: A\lambda^+ B\}$ and $r(A) = \{B: A\rho^+ B\}$. The algorithm of Warshall (1962) can be used to calculate λ^+, ρ^+. It thus follows that the sets $l(A)$, $r(A)$ can be easily computed for each A. These ideas are not pursued in detail as they are not central to this paper.

2.3. (*Symmetrical*) *precedence relations and precedence grammars*
Let G be a grammar and $A, B \in V \cup \{\vdash, \dashv\}$. Then we define

(*a*) $A \overset{s}{=} B$ if A, B occur adjacently (with A preceding B) on the right hand side of a production or in an element of S.
(*b*) A $\overset{s}{<} B$ if there is some D such that $A \overset{s}{=} D$ and $B \in l(D)$.
(*c*) $A \overset{s}{>} B$ if there is some C such that $C \overset{s}{=} B$ and $A \in r(C)$.
(*d*) $A \overset{s}{><} B$ if $A \overset{s}{<} B$ does not hold and
$A \overset{s}{>} B$ does not hold and
there exist C, D such that
$C \overset{s}{=} D$ and $A \in r(C)$, $B \in l(D)$.

The relations $\overset{s}{=}$, $\overset{s}{<}$, $\overset{s}{>}$, $\overset{s}{><}$ are called the (symmetrical) *precedence relations* henceforth abbreviated to *precedence* relations. A grammar G is called a *precedence grammar* if at most one of the relations $A \overset{s}{=} B$, $A \overset{s}{<} B$, $A \overset{s}{>} B$, $A \overset{s}{><} B$ holds for any pair of symbols $A, B \in V \cup \{\vdash, \dashv\}$.

If more than one relation holds we write $A \overset{s}{*} B$. If no relation holds we write $A \overset{s}{.} B$.
The distinctive points to note here are:

1. The left hand sides of productions are *not* used in defining the relation $\overset{s}{=}$
2. Initial strings in S and right hand sides of production are treated in the same way in the definition of $\overset{s}{=}$.
3. The relations $A \overset{s}{<} B$, $A \overset{s}{><} B$ by definition cannot both hold. Similarly $A \overset{s}{<} B$, $A \overset{s}{><} B$ cannot both hold.

4. If $C \overset{s}{=} D$ and $A \in r(C)$ and $B \in l(D)$, all that can be asserted is that one of the relations $A \overset{s}{><} B$, $A \overset{s}{<} B$ or $A \overset{s}{>} B$ holds. See definition of $\overset{s}{><}$.

The reader familiar with the algebra of relations will realise that the precedence relations can be equivalently defined as follows. Let $A\bar{\rho}B$ be the relation $B\rho A$. Then:

(*a*) $\overset{s}{=}$ is defined as before
(*b*) $\overset{s}{<}$ is defined as $\overset{s}{=} \lambda^+$
(*c*) $\overset{s}{>}$ is defined as $\bar{\rho}^+ \overset{s}{=}$
(*d*) $\overset{s}{><}$ is defined as $(\bar{\rho}^+ \overset{s}{=} \lambda^+) \wedge (\neg \overset{s}{<}) \wedge (\neg \overset{s}{>})$

Thus by using the algorithm of Warshall (1962) to calculate λ^+, ρ^+, the precedence relations may be easily calculated. See also 2.2.

3. Precedence parsing
In this section we develop a two stack parsing method for $L(G)$ the language of a simple precedence grammar G. The parser will always reduce a sentential form to an element of S but it may loop on other strings in $\vdash V^* \dashv$. However if the grammar is context sensitive and loop free (2.0) the parsing process will always terminate.
The parser is developed by introducing the concept of a correctly delimited substring (3.0) and then showing that only properly delimited substrings may be used when reducing a sentential form to an initial string (3.1). This is then followed by a description of the parser (3.2).

3.0. *Correctly delimited substrings*
Let G be a precedence grammar and let $x = A_0A_1 \ldots A_nA_{n+1}$ where $A_0 = \vdash$, $A_{n+1} = \dashv$ and $A_i \in V$; $i = 1, 2, \ldots, n$. A substring $y = A_iA_{i+1} \ldots A_j$ of x where $0 \leqslant i \leqslant j \leqslant n + 1$ is said to be correctly delimited if the three following conditions hold.

1. $A_i \overset{s}{=} A_{i+1} \ldots \overset{s}{=} A_j$
2. $i = 0$ or $A_{i-1} \overset{s}{<} A_i$ or $A_{i-1} \overset{s}{><} A_i$
3. $j = n + 1$ or $A_j \overset{s}{>} A_{j+1}$ or $A_j \overset{s}{><} A_{j+1}$

The distinctive point to note here is that $y = x$ is not excluded.

3.1. *Theorem*
Let G be a simple precedence grammar and let x be a *sentential form* which is *not* in S. Then x may be reduced to an initial string in S by repeatedly reducing the correctly delimited substrings of x using the productions of P. Furthermore the correctly delimited substring may be reduced in any order sequentially or in parallel.

Proof:
There are three assertions required which once proved will establish the theorem.

Assertion 1:
If a substring of x is correctly delimited it will remain correctly delimited no matter what reductions are performed adjacent to it.

Assertion 2:
No substring z of x which overlaps a correctly delimited substring y and such that $z \neq y$ can be used to reduce x to an initial string in S.

Assertion 3:
Every sentential form has at least one correctly delimited substring.

Downloaded from https://academic.oup.com/comjnl/article/17/3/234/389303 by guest on 21 August 2022

Once Assertions 1, 2 and 3 have been proved it is clear that a correctly delimited substring can only be reduced in its entirety or not at all. Because every sentential form has at least one correctly delimited substring, it follows that by repeatedly reducing them x may be reduced to an initial element of S. In view of Assertion 2, the reductions may be performed in any order, sequentially or in parallel. Furthermore as G is simple (2.0) it has no duplicated r.h.s. in its productions. Therefore the reduction process is completely deterministic. All that remains now is to establish Assertions 1, 2 and 3.

In what follows we use the notation of (3.0) in proving Assertions 1, 2 and 3. The general method of proof in deriving Assertions 1 and 2 is proof by contradiction. We show that G is not a precedence grammar, i.e. more than one precedence relation holds if Assertions 1 and 2 are invalid. Assertion 3 is proved directly.

Proof of Assertion 1:

Let $y = A_i \ldots A_j$ be a properly delimited substring of the sentential form $x = A_0 A_1 \ldots A_n A_{n+1}$. Suppose some reduction is made adjacent to y. Without loss of generality we may assume that the reduction was performed to the right of y. (A similar argument may be applied if the reduction was performed to the left of y). Let us suppose then that $A_{j+1} A_{j+2} \ldots A_k$ was reduced to $B_1 B_2 \ldots B_m$ by the production $B_1 B_2 \ldots B_m \rightarrow A_{j+1} A_{j+2} \ldots A_k$. We have to show that $y = A_i \ldots A_j$ is properly delimited after the reduction takes place. We use proof by contradiction. Consider $w = A_0 A_1 \ldots A_j B_1 B_2 \ldots B_m A_{k+1} \ldots A_{n+1}$.

If y is not properly delimited in w

$$\text{then either } A_j \overset{s}{=} B_1 \text{ or } A_j \overset{s}{<} B_1$$

Now as $A_{j+1} \in l(B_1)$ it follows that $A_j \overset{s}{<} A_{j+1}$. This would mean y is not properly delimited in x. Contradiction. This establishes Assertion 1.

Proof of Assertion 2:

We use proof by contradiction.

Suppose $z = A_h \ldots A_k$ where $h \leqslant k$ is a substring of a sentential form x which overlaps a correctly delimited substring y where $z \neq y$. Then clearly either $i \leqslant h \leqslant j$ or $i \leqslant k \leqslant j$. As $z \neq y$ $i = h$ and $k = j$ is impossible. Thus at least one of the following four cases hold.

1. $i \leqslant h$ and $k < j$
2. $h < i$ and $k \leqslant j$
3. $h < i \leqslant k$
4. $h \leqslant i < k$

Now suppose x may be reduced by the production $C_1 C_2 \ldots C_m \rightarrow z$ and that this reduction can be used to reduce x to an initial string in S. We shall derive a contradiction in each of cases 1 to 4 above.

Case 1:

As $A_i \ldots A_j$ is properly delimited it follows $A_k \overset{s}{=} A_{k+1}$. Consider the string after the reduction has taken place, i.e. examine

$$v = A_0 A_1 \ldots A_{h-1} C_1 C_2 \ldots C_m A_{k+1} \ldots A_n A_{n+2}\,.$$

Consider the relation holding between C_m and A_{k+1}.

If $C_m \overset{s}{=} A_{k+1}$ then as $A_k \in r(C_m)$, $A_k \overset{s}{>} A_{k+1}$.

Contradicting $A_k \overset{s}{=} A_{k+1}$.

If $C_m \overset{s}{>} A_{k+1}$ then as $A_k \in r(C_m)$, $A_k \overset{s}{>} A_{k+1}$.

This contradicts $A_k \overset{s}{=} A_{k+1}$.

If $C_m \overset{s}{<} A_{k+1}$ or $C_m \overset{s}{><} A_{k+1}$ then there exists X, Y such that $X \overset{s}{=} Y$ where C_m is X or $C_m \in r(X)$ and $A_{k+1} \in l(Y)$.

(see 2.2.—Definitions of $\overset{s}{<}$ and $\overset{s}{><}$).

As $A_k \in r(C_m)$ it follows that for the same X, Y we have $A_k \in r(X)$, $A_{k+1} \in l(Y)$ and $X = Y$. Thus either $A_k \overset{s}{<} A_{k+1}$ or $A_k \overset{s}{>} A_{k+1}$ or $A_k \overset{s}{><} A_{k+1}$. (see 2.2.—Definition of $\overset{s}{><}$).

This contradicts $A_k \overset{s}{=} A_{k+1}$.

Case 2:

This proved analogously to 1.

Case 3:

Suppose $h < i \leqslant k$. As $A_i \ldots A_j$ is correctly delimited and $h < i$ it follows that $A_{i-1} \overset{s}{<} A_i$ or $A_{i-1} \overset{s}{><} A_i$. But as $A_h \ldots A_k$ occurs on the r.h.s. of a production and $h < i \leqslant k$ it follows that $A_{i-1} \overset{s}{=} A_i$ which is a contradiction.

Case 4:

This is proved analagously to 3. This establishes Assertion 2.

Proof of Assertion 3:

One of the many ways of finding a correctly delimited substring is as follows:

1. Scan string from left to right.
2. Stop at the first occurrence of the relation $\overset{s}{>}$ or $\overset{s}{><}$ between adjacent symbols. Failing that stop at the end of the string. Clearly before stopping the only precedence relations encountered must be $\overset{s}{=}$ or $\overset{s}{<}$.
3. Now scan backwards from the position found in 2. and stop at the first occurrence of $\overset{s}{<}$. Failing that stop at the front of the string.

Clearly the substring so identified must be correctly delimited.

This establishes Assertion 3 and completes the proof of the Theorem 3.1.

3.2. *The two stack parser*

We shall describe a parser which identifies the leftmost correctly delimited substring and reduces it (where possible) by a production of P. Two stacks are used to identify the desired substring. It should be noted that this is just one possible order in which reductions may be performed since by Theorem 3.1 the reductions may be performed in any order sequentially or in parallel. As Theorem (3.1) deals with sentential forms only the problem of how to prevent the parser from looping when attempting to parse a non sentential form still remains. In general the problem of detecting and removing loops in a phrase structure grammar is an unsolvable problem as this problem is equivalent to the halting problem for Turing machines; see Hopcroft and Ullman (1969). However, if the grammar is a precedence context sensitive grammar which is loop free the reductions cannot be repeated indefinitely as the length of the string is never increased by performing a reduction.

The parser uses two stacks and a typical configuration of the stacks will be denoted 'x y' where x shall be called the left stack and y the right stack. The top of the left stack is the last symbol of x, the top of the right stack is the first symbol of y. Loosely speaking x denotes the part of the string that has been scanned and y denotes the part of the string remaining to be scanned. The operations required to describe the parser are the following.

1. Operations TOPL and TOPR giving the value of the top of the left and right stack respectively. We define TOPL $= e$ if the left stack is empty, TOPR $= e$ if the right stack is empty.

Downloaded from https://academic.oup.com/comjnl/article/17/3/234/389303 by guest on 21 August 2022

2. An operation REDUCE which reduces the correctly delimited substring found in the left stack by a production. The correctly delimited substring is deleted from the left stack and the string it is reduced to is placed on the right stack. If no reduction is possible *the parse fails.*

The configurations required for starting and terminating the parse are:

1. The initial configuration is '$e\,v$' where $v \in |\!\!-V^*\!\!-\!|$ is the string to be parsed and e is the null string.
2. The final or accepting configurations are '$s\,e$' where s ranges over the set S and e is the null string.

The parser will now be described using **'while do end' 'if then else'** and **'begin end'** notation.

begin
  while *the configuration is not final*
  do if $TOPL = e$ *or* $TOPL \overset{s}{=} TOPR$ *or* $TOPL \overset{s}{<} TOPR$
    then *move TOPR to the left stack*
    else if $TOPR = e$ *or* $TOPL \overset{s}{>} TOPR \overset{s}{><} TOPR$
      then *REDUCE* (if no reduction possible parse fails)
      else *parse fails.*
    The configuration is final—parse found.
  end

This completes the development of the theory of precedence parsing for phrase structure grammars. We illustrate these ideas with two examples.

Example 1
Let G_1 be a grammar where

$$V_1 = \{a, b, c, d, A, D, Y, Z\};\ T_1 = \{a, b, c, d\};$$
$$S_1 = \{|\!\!-AbcD\!\!-\!|\} \text{ and } P_1$$

has productions

$$bc \to YbcZ,$$
$$bY \to Yb, \quad Zc \to cZ,$$
$$aY \to aAb, \quad Zd \to cDd,$$
$$A \to a, \quad D \to d,$$

In this case $L(G_1) = \{a^n b^n c^n d^n : n \geqslant 1\}$ which is not a context free language.

The salient steps in the derivation of $|\!\!-a^n b^n c^n d^n\!\!-\!|$; $n = 1, 2, 3$ from the initial string $|\!\!-Abc\,D\!\!-\!|$ are shown below

$$|\!\!-AbcD\!\!-\!| \to {}^*|\!\!-abcd\!\!-\!|$$
$$\to |\!\!-aYbcZd\!\!-\!| \to {}^*|\!\!-aAbbccDd\!\!-\!|$$
$$\to {}^*|\!\!-aabbccdd\!\!-\!|$$
$$\to |\!\!-aabYbcZcdd\!\!-\!| \to {}^*|\!\!-aaYbbccZdd\!\!-\!|$$
$$\to {}^*|\!\!-aaAbbbcccDdd\!\!-\!| \to {}^*|\!\!-aaabbbcccddd\!\!-\!|$$

G_1 is in fact a precedence grammar, see **Tables 1 and 2** for the tables of left and right sets, and the precedence matrix.

The behaviour of the two stack parser in accepting and rejecting specimen strings is shown in **Figs. 1 and 2.** To help the reader to identify correctly delimited substrings we have inserted the precedence relations $<, >, ><$ on the left stack. The parser *does not* require the stacking of these relations.

Table 1 The left and right sets of G₁

X	$l(X)$	$r(X)$
a	$\{a\}$	$\varnothing$
b	$\{Y\}$	$\varnothing$
c	$\varnothing$	$\{Z\}$
d	$\varnothing$	$\{d\}$
A	$\{a\}$	$\{a\}$
D	$\{d\}$	$\{d\}$
Y	$\varnothing$	$\{b\}$
Z	$\{c\}$	$\varnothing$
$\vdash$	$\varnothing$	$\varnothing$
$\dashv$	$\varnothing$	$\varnothing$

Table 2 The precedence matrix of G₁

	a	b	c	d	A	D	Y	Z	$\vdash$	$\dashv$
a	$\overset{s}{<}$	$\overset{s}{>}$			$\overset{s}{=}$		$\overset{s}{><}$			
b		$\overset{s}{>}$	$\overset{s}{=}$				$\overset{s}{><}$			
c			$\overset{s}{<}$	$\overset{s}{<}$		$\overset{s}{=}$		$\overset{s}{=}$		
d				$\overset{s}{>}$						$\overset{s}{>}$
A		$\overset{s}{=}$					$\overset{s}{<}$			
D				$\overset{s}{=}$						$\overset{s}{=}$
Y		$\overset{s}{=}$					$\overset{s}{<}$			
Z			$\overset{s}{><}$	$\overset{s}{><}$		$\overset{s}{>}$		$\overset{s}{>}$		
$\vdash$	$\overset{s}{<}$				$\overset{s}{=}$					
$\dashv$										

e	$\vdash aaabbbcccddd\dashv$
$\vdash < a < a < a >$	$bbbcccddd\dashv$
$\vdash < a < a$	$Abbbcccddd\dashv$
$\vdash < a < aAb >$	$bbcccddd\dashv$
$\vdash < a$	$aYbbcccddd\dashv$
$\vdash < a < a ><$	$Ybbcccddd\dashv$
$\vdash < a$	$AYbbcccddd\dashv$
$\vdash < aA < Yb >$	$bcccddd\dashv$
$\vdash < aA$	$bYbcccddd\dashv$
$\vdash < aAb ><$	$Ybcccddd\dashv$
$\vdash$	$aYYbcccddd\dashv$
$\vdash < a ><$	$YYbcccddd\dashv$
$\vdash$	$AYYbcccddd\dashv$
$\vdash A < Y < Ybc < c < c < d >$	$dd\dashv$
$\vdash A < Y < Ybc < c < c$	$Ddd\dashv$
$\vdash A < Y < Ybc < c < cDd >$	$d\dashv$
$\vdash A < Y < Ybc < c$	$Zdd\dashv$
$\vdash A < Y < Ybc < cZ ><$	$dd\dashv$
$\vdash A < Y < Ybc$	$Zcdd\dashv$
$\vdash A < Y < YbcZ ><$	$cdd\dashv$
$\vdash A < Y$	$bccdd\dashv$
$\vdash A < Ybc < c < d >$	$d\dashv$
$\vdash A < Ybc < c$	$Dd\dashv$
$\vdash A < Ybc < cDd >$	$\dashv$
$\vdash A < Ybc$	$Zd\dashv$
$\vdash A < YbcZ ><$	$d\dashv$
$\vdash A$	$bcd\dashv$
$\vdash Abc < d >$	$\dashv$
$\vdash Abc$	$D\dashv$
$\vdash AbcD\dashv$	e

Fig. 1 Parse of $\vdash aaabbbcccddd\dashv$ **in** G_1

Example 2:
Let G_2 be a grammar where

$$V_2 = \{a, b, c, A, B\};\ T_2 = \{a, b, c\};\ S_2 = \{|\!\!-c\!\!-\!|\}$$

P_2 has productions

$$c \to acA,$$
$$c \to bcB,$$
$$A\!\!-\!| \to a\!\!-\!|, \quad B\!\!-\!| \to b\!\!-\!|,$$
$$Aa \to aA, \quad Bb \to bB,$$
$$Ab \to bA, \quad Ba \to aB.$$

$L(G_2) = \{tct : t \in \{a, b\}^*\}$ which is not a context free language. G_2 is in fact a precedence grammar. See **Tables 3 and**

Downloaded from https://academic.oup.com/comjnl/article/17/3/234/389303 by guest on 21 August 2022

e	$\vdash aaabbbccdd \dashv$
$\vdash < a < a < a >$	$bbbccdd \dashv$
$\vdash < a < a$	$Abbbccdd \dashv$
$\vdash < a < aAb >$	$bbccdd \dashv$
$\vdash < a$	$aYbbccdd \dashv$
$\vdash < a < a > <$	$Ybbccdd \dashv$
$\vdash < a$	$AYbbccdd \dashv$
$\vdash < aA < Yb >$	$bccdd \dashv$
$\vdash < aA$	$bYbcdd \dashv$
$\vdash < aAb >$	$Ybcdd \dashv$
$\vdash$	$aYYbcdd \dashv$
$\vdash < a >$	$YYbcdd \dashv$
$\vdash$	$AYYbcdd \dashv$
$\vdash A < Y < Ybc < d >$	$d \dashv$
$\vdash A < Y < Ybc$	$Dd \dashv$
$\vdash A < Y < YbcDd >$	$\dashv$
Parse Fails here	

Fig. 2 Failure of the parse of $\vdash aaabbbccdd \dashv$ in G_1

4 for the tables of left, right sets and the precedence matrix.

The behaviour of the two stack parser in accepting a specimen string is shown in **Fig. 3.** To help the reader identify the correctly delimited substrings, we have inserted the precedence relations $<$, $>$, $><$ on the left stack. The parser *does not* require the stacking of these relations.

4. Conversion to precedence form

In this section we define context expansions (4.0) and use them to transform any phrase structure grammar into precedence form (4.1).

The original notation of an expansion was introduced by Learner and Lim (1970) to transform any context free grammar to Wirth Weber precedence form. Lim (1972) defines various types of expansions (including context expansions) on context free grammars and discusses various transformational algorithms. The approach we take here is not dissimilar to the approach of Lim (1972).

4.0. (*Context*) *expansion*

Let G be a grammar and $A \in V$. A (context) *expansion* of A (henceforth abbreviated to *expansion*), transforms a grammar G into G' while keeping $L(G) = L(G')$. It adds at most three new symbols A_L, A_R, A_M to V and alters S and P in the following manner.

1. All elements which occur on the r.h.s. of a production *or* in S and which have length greater than one are examined. Let q be a typical element of this type.
2. (*a*) If the leftmost symbol of q is A it is replaced by A_L.
   (*b*) If the rightmost symbol of q is A it is replaced by A_R.
   (*c*) All other occurrences of A in q are replaced by A_M.
3. If A_Q; $Q = L, R, M$; has replaced A in Step 2, then the new symbols A_Q are added to V and the productions $A_Q \rightarrow A$ are added to P.

G' is the resulting grammar after 1, 2, 3 have been carried out.

The distinctive points to note here are

(*a*) Expansions are defined for elements of V only. They are not defined on the endmarkers $\vdash$ and $\dashv$.
(*b*) The left hand sides of the productions are not affected by expansions. Neither are any right hand sides of length one.
(*c*) The right hand sides of productions and elements of S are treated in exactly the same way; see 1, 2 above.
(*d*) $L(G') = L(G)$.

Table 3 Left and right sets of G_2

X	$l(X)$	$r(X)$
a	$\varnothing$	$\{A, B\}$
b	$\varnothing$	$\{A, B\}$
c	$\{a, b\}$	$\{A, B\}$
A	$\{a, b\}$	$\varnothing$
B	$\{a, b\}$	$\varnothing$
$\vdash$	$\varnothing$	$\varnothing$
$\dashv$	$\varnothing$	$\{\dashv\}$

Table 4 Precedence matrix of G_2

	a	b	c	A	B	$\vdash$	$\dashv$
a	$\overset{s}{<}$	$\overset{s}{<}$	$\overset{s}{=}$	$\overset{s}{=}$	$\overset{s}{=}$		$\overset{s}{=}$
b	$\overset{s}{<}$	$\overset{s}{<}$	$\overset{s}{=}$	$\overset{s}{=}$	$\overset{s}{=}$		$\overset{s}{=}$
c	$\overset{s}{<}$	$\overset{s}{<}$		$\overset{s}{=}$	$\overset{s}{=}$		$\overset{s}{=}$
A	$\overset{s}{><}$	$\overset{s}{><}$	$\overset{s}{>}$	$\overset{s}{>}$	$\overset{s}{>}$		$\overset{s}{>}$
B	$\overset{s}{><}$	$\overset{s}{><}$	$\overset{s}{>}$	$\overset{s}{>}$	$\overset{s}{>}$		$\overset{s}{>}$
$\vdash$	$\overset{s}{<}$	$\overset{s}{<}$	$\overset{s}{=}$				
$\dashv$							

e	$\vdash aabcaab \dashv$
$\vdash < a < a < bc < a < a < b \dashv$	e
$\vdash < a < a < bc < a < a$	$B \dashv$
$\vdash < a < a < bc < a < aB >$	$\dashv$
$\vdash < a < a < bc < a$	$Ba \dashv$
$\vdash < a < a < bc < aB > <$	$a \dashv$
$\vdash < a < a < bc$	$Baa \dashv$
$\vdash < a < a < bcB >$	$aa \dashv$
$\vdash < a < a$	$caa \dashv$
$\vdash < a < ac < a < a \dashv$	e
$\vdash < a < ac < a$	$A \dashv$
$\vdash < a < ac < aA >$	$\dashv$
$\vdash < a < ac$	$Aa \dashv$
$\vdash < a < acA > <$	$a \dashv$
$\vdash < a$	$ca \dashv$
$\vdash < ac < a \dashv$	e
$\vdash < ac$	$A \dashv$
$\vdash < acA >$	$\dashv$
$\vdash$	$c \dashv$
$\vdash c \dashv$	e

Fig. 3 Parse of $\vdash aabcaab \dashv$ in G_2

4.1. *The conversion algorithm*

Let G be a grammar. Perform expansions on all symbols $A \in V$ involved in precedence clashes and let the resulting grammar be G'. Then G' is a precedence grammar and $L(G) = L(G')$.

Proof

The symbols of V' may be grouped into three classes:

(*a*) Old symbols which have been expanded. These will be denoted by A, B.
(*b*) New symbols created by expansions. These will be denoted by A_Q, B_Q; $Q = L, R, M$.
(*c*) Old symbols which have not been expanded. These will be denoted by C, D.

Let X be any symbol in V' and let us use the classification introduced above. To show G is a precedence grammar we need only show Assertions 1 to 4 below.

Assertion 1

There are no clashes of the form $A \overset{s}{*} X$ or $X \overset{s}{*} A$ in G'.

Downloaded from https://academic.oup.com/comjnl/article/17/3/234/389303 by guest on 21 August 2022

Assertion 2

There are no clashes in G' of the form $A_Q \overset{s}{*} X$ or $X \overset{s}{*} A_Q$; $Q = L, R, M$.

Assertion 3

There are no clashes of the form $C * X$ or $X * C$ in G'.

Assertion 4

The endmarkers are not involved in any precedence clashes in G'.

Clearly once these have been established then G' will be shown to be free from precedence clashes and thus be a precedence grammar. Furthermore $L(G) = L(G')$ as expansions do not alter the language of G.

Proof of Assertion 1

We show only $A \overset{s}{*} X$ is impossible as the proof of $X \overset{s}{*} A$ impossible is analogous.

After expansion, A occurs on the right hand sides of productions of the form $w \to A$ only. Neither does A occur in any string in S. Thus $A \overset{s}{=} X$ is impossible for any X. Similarly if $A \overset{s}{<} X$ held then $X \in l(Y)$ and $A \overset{s}{=} Y$ for some $Y \in V$. But $A \overset{s}{=} Y$ cannot hold so $A \overset{s}{<} X$ cannot hold. Thus the only possibilities remaining are $A \overset{s}{>} X$ and $A \overset{s}{><} X$ and both of these cannot hold by definition of $\overset{s}{><}$.

To show $X \overset{s}{*} A$ is impossible analogous reasoning is used. It turns out that $X \overset{s}{<} A$ and $X \overset{s}{><} A$ are the only relations which can hold. Again both of these cannot hold by definition of $\overset{s}{><}$.

This completes the proof of Assertion 1.

Proof of Assertion 2

To show $A_Q \overset{s}{*} X$, $X \overset{s}{*} A_Q$ are impossible.
Two cases are considered:

(*a*) $A_R \overset{s}{*} X$, $X \overset{s}{*} A_L$ impossible

(*b*) $A_L \overset{s}{*} X$, $A_M \overset{s}{*} X$ impossible and $X \overset{s}{*} A_R$, $X \overset{s}{*} A_M$ impossible.

Case (*a*):

We only show $A_R \overset{s}{*} X$ impossible as the proof of $X \overset{s}{*} A_L$ impossible is analogous.

As A_R occurs only as the rightmost symbol of a production or of an initial element it follows that $A_R \overset{s}{=} X$ is impossible for any X. Similarly if $A_R \overset{s}{<} X$ held then $X \in l(Y)$ and $A_R \overset{s}{=} Y$ for some Y. But $A_R \overset{s}{=} Y$ is impossible so $A_R \overset{s}{<} X$ is impossible. The only remaining possibilities are $A_R \overset{s}{>} X$ or $A_R \overset{s}{><} X$ and both of these cannot hold.

To show $X \overset{s}{*} A_L$ impossible a similar argument shows that $X \overset{s}{<} A_L$ or $X \overset{s}{><} A_L$ are the only precedence relations which can hold. Again both of these cannot hold. This completes case (*a*).

Case (*b*):

We only show $A_L \overset{s}{*} X$, $A_M \overset{s}{*} X$ impossible as the proof of $X \overset{s}{*} A_R$, $X \overset{s}{*} A_M$ is analogous. Consider A_Q when $Q = L$ or M. Examine the possibilities for X. Three subcases arise:

(i) X is an old symbol which has been expanded
(ii) X is a new symbol
(iii) X is an old symbol which has not been expanded.

Subcase (*i*)

If X is an old symbol which has been expanded then Assertion 1 applies and $A_Q \overset{s}{*} X$ is impossible $Q = L, M$. This completes subcase (i).

Subcase (*ii*)

If X is a new symbol it must have the form B_L or B_R or B_M.

If X is B_L then case (*a*) applies and $Y \overset{s}{*} B_L$ is impossible for any Y and so in particular $A_Q * B_L$, i.e. $A_Q * X$ is impossible for $Q = L, M$.

Otherwise X is B_0 where $0 = R$ or M. As $A_Q \notin r(Y)$ for any Y; $Q = L, M$ it follows $A_Q \overset{s}{>} Z$. $A_Q \overset{s}{><} Z$ is impossible for any Z.

Similarly as $B_0 \notin l(Y)$ for any Y; $0 = R, M$ it follows $Z \overset{s}{<} B_0$ and $Z \overset{s}{><} B_0$ is impossible for any Z. Combining these two facts it follows that $A_Q \overset{s}{=} B_0$ is the only relation possible. Thus $A_Q \overset{s}{*} B_0$ is impossible, i.e. $A_Q \overset{s}{*} X$ is impossible. This completes subcase (ii).

Subcase (*iii*)

In G' we must show $A_Q \overset{s}{*} X$ impossible; $Q = L, M$ where X is an old symbol which has not been expanded. As X has not been expanded and as all symbols involved in precedence clashes were expanded it follows that X was involved in no precedence clash in G. So in G the following possibilities arise:

If $A \overset{s}{.} X$ in G, i.e. no precedence relation holds
then $A_Q \overset{s}{.} X$ in G'
If $A \overset{s}{=} X$ in G
then $A_Q \overset{s}{=} X$ is the only relation which can hold in G'.
If $A \overset{s}{<} X$ in G then G; $A \overset{s}{=} Y$ and $X \in l(Y)$ for some Y.
Thus in G' the only possibility is $X \in l(Y)$ where $A_Q \overset{s}{=} Y$ or alternatively if Y has been expanded then $X \in l(Y_0)$ where $A_Q \overset{s}{=} Y_0$; $0 = L$ or R or M. Thus $A_Q \overset{s}{<} X$ is the only possibility in G'.

If $A \overset{s}{>} X$ or $A \overset{s}{><} X$ in G then in the grammar G there exist Y and Z such that $Y \overset{s}{=} Z$ and $A \in r(Y)$ and X is Z or $X \in l(Z)$. (This is by definition of $\overset{s}{>}$ and $\overset{s}{><}$). But in G' $A_Q \notin r(W)$ for any W; $Q = L, M$. Thus $A_Q \overset{s}{.} X$ in G'.
This completes subcase (iii) and case (*b*).
Thus Assertion 2 is now proved.

Proof of Assertion 3

We show only $C \overset{s}{*} X$ is impossible. X is any symbol in V' and C is an unexpanded symbol. The proof of $X \overset{s}{*} C$ impossible is analogous.

Consider the possibilities for X.

(*a*) If X is an old symbol which has been expanded then assertion 1 applies and $C \overset{s}{*} X$ is impossible.

(*b*) If X is a new symbol then assertion 2 applies and $C \overset{s}{*} X$ is impossible.

Downloaded from https://academic.oup.com/comjnl/article/17/3/234/389303 by guest on 21 August 2022

Table 5 Left and right sets of G_3

X	$l(X)$	$r(X)$
a	$\{a, b\}$	$\varnothing$
b	$\varnothing$	$\{a, b\}$
$\vdash$	$\varnothing$	$\varnothing$
$\dashv$	$\varnothing$	$\varnothing$

Table 6 Precedence matrix of G_3

	a	b	$\vdash$	$\dashv$
a	$\overset{s}{*}$	$\overset{s}{*}$		$\overset{s}{>}$
b	$\overset{s}{*}$	$\overset{s}{*}$		$\overset{s}{*}$
$\vdash$	$\overset{s}{*}$	$\overset{s}{<}$		
$\dashv$				

Table 7 Left and right sets of G'_3

X	$l(X)$	$r(X)$
a	$\{a_L, b_L, a, b\}$	$\varnothing$
a_L, a_R, a_M	$\{a_L, b_L, a, b\}$	$\{a\}$
b	$\varnothing$	$\{a_R, b_R, a, b\}$
b_L, b_R, b_M	$\{b\}$	$\{a_R, b_R, a, b\}$
$\vdash$	$\varnothing$	$\varnothing$
$\dashv$	$\varnothing$	$\varnothing$

(*c*) If X is an old symbol which has not been expanded then both C and X are not involved in any clashes in G. (Recall that all symbols involved in clashes are expanded). Hence clearly C and X are not involved in any clashes in G'.

This completes the proof of assertion 3.

Proof of Assertion 4

To show that endmarkers are not involved in any clash.

Firstly in G' $\vdash \overset{s}{=} \dashv$, $\dashv \overset{s}{\cdot} \vdash$, $\vdash \overset{s}{\cdot} \vdash$, $\dashv \overset{s}{\cdot} \dashv$ are the only possibilities. This is because $\vdash$, $\dashv$ only occur as the leftmost (rightmost) symbol in initial elements or on the r.h.s. of a production.

Secondly let X be any symbol in V'. By using similar arguments to those used in the proofs of assertions 1, 2 and 3 the following are easily shown.

(*a*) If X is an old symbol which has been expanded then $\vdash \overset{s}{<} X$, $X \overset{s}{>} \dashv$ are the only possibilities in G'.

(*b*) If X is a new symbol, i.e. either A_L, A_R or A_M then $\vdash \overset{s}{<} A_L$, $\vdash \overset{s}{\cdot} A_R$, $\vdash \overset{s}{=} A_M$ and $A_R \overset{s}{>} \dashv$, $A_L \overset{s}{\cdot} \dashv$, $A_M \overset{s}{=} \dashv$ are the only possibilities in G'.

(*c*) If X is an old symbol which has not been expanded then X did not clash with the endmarkers in G. Thus X cannot clash with the endmarkers in G'.

In all cases therefore, the endmarkers are not involved in clashes. This completes the proof of Assertion 4 and establishes the correctness of Conversion Algorithm 4.1.

We illustrate these ideas with two examples.

Example 3

Let G be a grammar where

$$V_3 = T_3 = \{a, b\} \;\; S_3 = \{\vdash ab \dashv\}\,.$$

Let P_3 have productions

$$ab \to aabb,\; ab \to ba\,.$$

$L(G_3) = \{t \in T^*$: the number of a's and b's in t are equal$\}$ G_3 is not a precedence grammar. See **Tables 5 and 6.** All symbols in V_3 are involved in precedence clashes. Expanding them produces the grammar G'_3 where

$$V'_3 = \{a, b, a_L, a_R, a_M, b_L, b_R, b_M\};\; T'_3 = \{a, b\}$$
$$S'_3 = \{\vdash a_M\, b_M \dashv\}$$

P'_3 has productions

Table 8 Precedence matrix of G'_3

	a	a_L	a_R	a_M	b	b_L	b_R	b_M	$\vdash$	$\dashv$
a	$\overset{s}{>\!<}$	$\overset{s}{>\!<}$	$\overset{s}{>}$	$\overset{s}{>}$	$\overset{s}{>\!<}$	$\overset{s}{>\!<}$	$\overset{s}{>}$	$\overset{s}{>}$		$\overset{s}{>}$
a_L	$\overset{s}{<}$	$\overset{s}{<}$		$\overset{s}{=}$	$\overset{s}{<}$	$\overset{s}{<}$				
a_R	$\overset{s}{>\!<}$	$\overset{s}{>\!<}$	$\overset{s}{>}$		$\overset{s}{>\!<}$	$\overset{s}{>\!<}$	$\overset{s}{>}$			$\overset{s}{>}$
a_M					$\overset{s}{<}$			$\overset{s}{=}$		
b	$\overset{s}{>\!<}$	$\overset{s}{>\!<}$	$\overset{s}{>}$		$\overset{s}{>\!<}$	$\overset{s}{>\!<}$	$\overset{s}{>}$			$\overset{s}{>}$
b_L	$\overset{s}{<}$	$\overset{s}{<}$	$\overset{s}{=}$		$\overset{s}{<}$	$\overset{s}{<}$				
b_R	$\overset{s}{>\!<}$	$\overset{s}{>\!<}$	$\overset{s}{>}$		$\overset{s}{>\!<}$	$\overset{s}{>\!<}$	$\overset{s}{>}$			$\overset{s}{>}$
b_M					$\overset{s}{<}$		$\overset{s}{=}$			$\overset{s}{=}$
$\vdash$	$\overset{s}{<}$	$\overset{s}{<}$		$\overset{s}{=}$	$\overset{s}{<}$	$\overset{s}{<}$				
$\dashv$										

Table 9 Left and right sets of G_4

X	$l(X)$	$r(X)$
a	$\{a, b\}$	$\{a, c\}$
b	$\{b\}$	$\{b\}$
c	$\{c\}$	$\{c\}$
$\vdash$	$\varnothing$	$\varnothing$
$\dashv$	$\varnothing$	$\varnothing$

Table 10 Precedence matrix of G_4

	a	b	c	$\vdash$	$\dashv$
a			$\overset{s}{*}$		$\overset{s}{*}$
b	$\overset{s}{*}$	$\overset{s}{<}$	$\overset{s}{*}$		
c		$\overset{s}{*}$	$>$		$\overset{s}{>}$
$\vdash$	$\overset{s}{*}$	$\overset{s}{<}$			
$\dashv$					

Table 11 Left and right sets of G'_4

X	$l(X)$	$r(X)$
$a;\, a_L, a_R, a_M$	$\{a_L, b_L, a, b\}$	$\{a_R, c_R, a, c\}$
b, b_L, b_R, b_M	$\{b_L, b\}$	$\{b_R, b\}$
c, c_L, c_R, c_M	$\{c_L, c\}$	$\{c_R, c\}$
$\vdash$	$\varnothing$	$\varnothing$
$\dashv$	$\varnothing$	$\varnothing$

Downloaded from https://academic.oup.com/comjnl/article/17/3/234/389303 by guest on 21 August 2022

$$ab \to a_L a_M b_M b_R,\ ab \to b_L a_R,$$
$$a_L \to a,\ a_R \to a,\ a_M \to a$$
$$b_L \to a,\ b_R \to a,\ b_M \to a$$

G_3' is a precedence grammar as is evident from **Tables 7 and 8.**

Example 4
Let G_4 be a grammar where

$$V_4 = T_4 = \{a, b, c\};\ S_4 = \{|\!\!-a-\!\!|\}$$

P_4 has productions

$$a \to ba,$$
$$a \to ac,$$
$$b \to bcb,\ c \to cbc.$$

It is obvious that G_4 can be transformed into a context free grammar if this is required. G_4 is not a precedence grammar as is evident from **Tables 9 and 10.** All symbols in V_4 are involved in precedence clashes. Expanding them produces the grammar G'_4 where

$$V_4' = \{a, b, c, a_L, a_R, a_M, b_L, b_R, b_M, c_L, c_R, c_M\};$$
$$T_4' = \{a, b, c\}\ S_4' = \{|\!\!-a_M-\!\!|\}$$

P'_4 has productions

$$a \to b_L a_R,$$
$$a \to a_L c_R,$$
$$b \to b_L c_M b_R, \qquad c \to c_L b_M c_R,$$
$$a_L \to a, \quad a_R \to a, \quad a_M \to a,$$
$$b_L \to b, \quad b_R \to b, \quad b_M \to b,$$
$$c_L \to c, \quad c_R \to c, \quad c_M \to c.$$

G_4' is a precedence grammar as is evident from **Tables 11 and 12.**

5. Conclusion
The extension of the theory of precedence to general phrase structure grammars provides a useful and elegant tool for the definition and parsing of languages. This together with the algorithm for converting a p.s.g. to precedence form should provide a good method for the design and implementation of syntax analysers.

6. Acknowledgement
The author wishes to thank A. L. Lim for the many helpful discussions on the ideas presented in this paper.

Table 12 Precedence matrix of G_4'

	a	a_L	a_R	a_M	b	b_L	b_R	b_M	c	c_L	c_R	c_M	$\|\!-$	$-\!\|$
a									$\overset{s}{><}$	$\overset{s}{><}$	$\overset{s}{>}$			$\overset{s}{>}$
a_L									$\overset{s}{<}$	$\overset{s}{<}$	$\overset{s}{=}$			
a_R									$\overset{s}{><}$	$\overset{s}{><}$	$\overset{s}{>}$			$\overset{s}{>}$
a_M														$\overset{s}{=}$
b	$\overset{s}{><}$	$\overset{s}{><}$	$\overset{s}{>}$		$\overset{s}{><}$	$\overset{s}{><}$			$\overset{s}{><}$	$\overset{s}{><}$	$\overset{s}{>}$	$\overset{s}{>}$		
b_L	$\overset{s}{<}$	$\overset{s}{<}$	$\overset{s}{=}$		$\overset{s}{<}$	$\overset{s}{<}$			$\overset{s}{<}$	$\overset{s}{<}$		$\overset{s}{=}$		
b_R	$\overset{s}{><}$	$\overset{s}{><}$	$\overset{s}{>}$		$\overset{s}{><}$	$\overset{s}{><}$			$\overset{s}{><}$	$\overset{s}{><}$	$\overset{s}{>}$	$\overset{s}{>}$		
b_M									$\overset{s}{<}$	$\overset{s}{<}$	$\overset{s}{=}$			
c					$\overset{s}{><}$	$\overset{s}{><}$	$\overset{s}{>}$	$\overset{s}{>}$	$\overset{s}{><}$	$\overset{s}{><}$	$\overset{s}{>}$			$\overset{s}{>}$
c_L					$\overset{s}{<}$	$\overset{s}{<}$		$\overset{s}{=}$						
c_R					$\overset{s}{><}$	$\overset{s}{><}$	$\overset{s}{>}$	$\overset{s}{>}$	$\overset{s}{><}$	$\overset{s}{><}$	$\overset{s}{>}$			$\overset{s}{>}$
c_M					$\overset{s}{<}$	$\overset{s}{<}$	$\overset{s}{=}$							
$\|\!-$	$\overset{s}{<}$	$\overset{s}{<}$		$\overset{s}{=}$	$\overset{s}{<}$	$\overset{s}{<}$								
$-\!\|$														

References

COLMERAUER, A. (1970). Total Precedence Relations, *JACM*, Vol. 17, No. 1, pp. 14-30.

FLOYD, R. W. (1963). Syntactic Analysis and Operator Precedence, *JACM*, Vol. 10, No. 3, pp. 316-333.

HOPCROFT, J. E., and ULLMAN, J. D. (1969). *Formal Languages and their relation to Automata*, New York—Addison Wesley.

LEARNER, A., and LIM, A. L. (1970). A Note on Transforming Grammars to Wirth-Weber Precedence Form, *Computer Journal*, Vol. 12, No. 2, pp. 142-144.

LIM, A. L. (1972). *Useful Algorithms for Generating Wirth-Weber Grammars for Context-free Languages*, CCD (Imperial College) Research Report 72/15.

WARSHALL, S. (1962). A Theorem on Boolean Matrices, *JACM*, Vol. 9, No. 1, pp. 11, 12.

WIRTH, N., and WEBER, H. (1966). Euler: A Generalisation of Algol and its Formal Definition, Parts I and II, CACM, Vol. 9, pp. 13-25 and 88-89.

Downloaded from https://academic.oup.com/comjnl/article/17/3/234/389303 by guest on 21 August 2022