On context resolvable grammars

Fred John Zamecnik Jr.

Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Part of the Computer Sciences Commons

Recommended Citation

https://lib.dr.iastate.edu/rtd/4521

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.
ZAMECNIK, Jr., Fred John, 1935-
ON CONTEXT RESOLVABLE GRAMMARS.
Iowa State University, Ph.D., 1971
Computer Science

University Microfilms, A XEROX Company, Ann Arbor, Michigan

THIS DISSERTATION HAS BEEN MICROFILMED EXACTLY AS RECEIVED
On context resolvable grammars

by

Fred John Zamecnik Jr.

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of
The Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa

1971
PLEASE NOTE:

Some Pages have indistinct print. Filmed as received.

UNIVERSITY MICROFILMS
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>PRECEDENCE GRAMMARS</td>
<td>7</td>
</tr>
<tr>
<td>CONTEXT RESOLVABLE GRAMMARS</td>
<td>18</td>
</tr>
<tr>
<td>ALGORITHMS AND EXAMPLES</td>
<td>32</td>
</tr>
<tr>
<td>DISCUSSION</td>
<td>51</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>54</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>56</td>
</tr>
</tbody>
</table>
INTRODUCTION

Overview

In recent years, many techniques for generating syntax directed compilers have evolved. Precedence analysis was used prior to formal language theory and the development of formal classes of grammars. Randell and Russell (11) translated arithmetic expressions via an algorithm which used a set of precedence values for operators to determine the evaluation sequence. Floyd (7) developed a very efficient analyzer for a class of grammars he defined as operator precedence grammars. The analyzer is constructed to recognize an input string without having to backup at some point and consider an alternate decision. Operator precedence relations however are defined only for terminal symbols and this restricts the types of grammars which can satisfy the definition.

Wirth and Weber (13) generalized the idea of precedence, to include a larger class of grammars than operator precedence, by defining precedence relations over the whole alphabet of symbols - both non-terminal and terminal. The precedence grammars also allow for a simpler parsing algorithm than Floyd's (7). While some practical programming languages exist which can be defined by precedence grammars, it would be desirable to increase the class of languages which can be parsed by a precedence type analyzer.

Work with formal grammars by Knuth (9), resulted in a class of grammars called LR(k), for which a parsing algorithm can be constructed to run without backup. Knuth's (9) work is significant because he shows that the LR(k) grammars define exactly those grammars which are deterministic (no
backup is required). A disadvantage of LR(k) parsing algorithms is that a great deal of information (state sets) must be used to determine a parse. In fact, the amount of space required for the state sets for large grammars can be prohibitive. A specific example shown by Earley (3) contains a grammar whose state sets grow in size exponentially with respect to the size of the grammar.

The size requirements for precedence analyzers however vary only linearly with respect to the grammar. If the class of languages definable by precedence grammars could be increased, the size requirement and efficient parsing algorithm of precedence grammars make them attractive candidates for compilers.

The basis of a precedence grammar is the precedence relations which are defined such that at most one precedence relation can hold between any two symbols. A non-precedence grammar contains symbol pairs for which more than one relation holds. This is called a precedence conflict.

If it were possible to modify the definition of precedence grammars to eliminate the conflicts or at least resolve the conflict, then the definition would provide for a larger class of definable grammars.

Preliminary Concepts

A considerable amount of notation exists in the field of formal grammars and the following definitions will serve to act as the standard for this thesis.
Definition 1.1 (ALPHABET)

Let $A$ be a finite non-empty set called an alphabet.

Definition 1.2 (STRING)

A string over an alphabet $A$ is a finite sequence of elements of $A$ which is formed by concatenating the elements of $A$.

Definition 1.3 (NULL STRING)

A string over the alphabet $A$ of zero length is called the null string and is denoted by $\phi$. A word is a non-null string of length $\geq 1$. The length of a word $a$ is denoted $|a|$.

Definition 1.4 ($A^*$ and $A^+$)

The set of all strings over the alphabet $A$, including the null string is denoted by $A^*$. $A^+$ is used to denote the set $A^* - \{\phi\}$.

Definition 1.5 (PRODUCTION)

A production is an ordered pair of strings $(x, u)$ over $A^*$ which is generally written as $x \rightarrow u$. $x$ is called the left part of the production and $u$ is called the right part.

Definition 1.6 (PHRASE-STRUCTURE GRAMMAR)

A phrase-structure grammar is a 4-tuple denoted $G = (V_N, V_T, P, S)$ where

1. $V_N$ is the non-terminal alphabet contained in $A$,
2. $V_T$ is the terminal alphabet contained in $A$.
3. $P$ is a finite set of productions over $A^*$,
4. $S$ is a unique starting symbol.
The terminal alphabet \( V_T \) are those elements of \( A \) which appear only on the right part of a production. The non-terminal alphabet \( V_N = A - V_T \). The starting symbol is \( S \) if and only if \( S \) is a member of \( V_N \) and \( S \) does not appear in the right part of any production.

**Definition 1.7 (CONTEXT-FREE GRAMMAR)**

A context-free grammar is a phrase-structure grammar with the following restrictions on the set \( P \):

1. \( |x| = 1 \)
2. \( \chi \) must be a member of \( V_N \).

Unless otherwise specified, the upper case alphabetic letters A thru Z will be used for non-terminal symbols in \( V_N \) and the lower case a to z will denote the terminal symbols of \( V_T \). The Greek letters \( \alpha \) thru \( \omega \) will be used to represent strings in \( A^* \).

**Definition 1.8 (DERIVATION)**

If \( Y \rightarrow \gamma \) is a production in \( P \) and \( \alpha, \omega \in A^* \) then it is said \( \alpha Y \omega \) directly derives \( \alpha \gamma \omega \) (denoted \( \alpha Y \omega \Rightarrow \alpha \gamma \omega \)).

If there are strings \( \alpha_1 \) in \( A^* \) such that \( \alpha_1 \Rightarrow \alpha_2 \Rightarrow \alpha_3 \ldots \alpha_{k-1} \Rightarrow \alpha_k \) then \( \alpha_k \) is derived from \( \alpha_1 \) and is written \( \alpha_1 \Rightarrow^* \alpha_k \).

**Definition 1.9 (REDUCTION)**

The inverse relation of a derivation is called a reduction and \( \alpha \gamma \omega \) directly reduces to \( \alpha Y \omega \).
Definition 1.10 (CANONICAL PARSE)

The parse of a string $a (a \in A^+)$ to the starting symbol $S$ which proceeds strictly left to right and reduces the leftmost part of the string as far as possible before proceeding further to the right is called a canonical parse.

Definition 1.11 (SENTENTIAL FORM)

A string $a$ is a sentential form of $A$ if there exists an element $S \in V_N$ and a sequence of direct derivations such that $S \Rightarrow^* a$.

Definition 1.12 (LANGUAGE)

The language generated by the grammar $G$ is defined as:

$$L(G) = \{ a | S \Rightarrow^* a \text{ and } a \in V_T^* \}.$$ 

Approach and Outline

The main emphasis of this thesis will concern itself with the modification of the definition of precedence grammars to eliminate precedence conflicts of certain classes of grammars; it will also define a method for resolving precedence conflicts for other classes of grammars. Grammars which satisfy this augmented definition will be called context resolvable grammars. In addition, the necessary generating algorithm for determining if a grammar is context resolvable will be presented. And then, the associated parsing algorithm will be developed. Finally, the generality of context resolvable grammars will be discussed.
Chapter 2 will present the previous work pertaining to precedence grammars. Both simple precedence grammars, and more general types will be discussed. Chapter 3 will introduce the modifications to the definition of precedence \((m,n)\) grammars and thereby define context resolvable grammars. The associated left and right contextual sets will be defined, which are part of the context resolvable grammar definition. Set relations of the various grammars will be presented to show the various subset relations.

In Chapter 4, the generating algorithm for context resolvable grammars is developed along with a modified precedence analyzer for context resolvable grammars. A section of illustrated examples will also be provided to better display various aspects of the formal definitions.
In this chapter the various types of precedence grammars will be presented. The presentation shows the original development of the area and various extensions which have occurred.

Simple Precedence Systems

Precedence concepts

Informally, precedence analysis is a process which reduces an input string by replacing the handle at each point. Given a string to parse, the sentential form at any stage is comprised of symbols in $A^*$. For any string of symbols $A_iA_{i+1}$, which do occur in a successful parse, and one or both are in the handle then they are related in three possible ways:

1. The symbol $A_i$ is the character immediately left of a handle, and $A_{i+1}$ is the leftmost symbol of that handle
2. The symbol $A_i$ and $A_{i+1}$ are adjacent symbols contained in a handle
3. The symbol $A_i$ is the rightmost symbol of a handle and $A_{i+1}$ is the symbol immediately to the right of that handle.

The relations which denote these three conditions will be called precedence relations. The symbols $<, =, >$ will be used to denote the individual precedence relations, and if at most one precedence relation holds between any ordered pair of symbols $(A_i, A_j)$, then these relations can be used to determine the handle.
The handle is that leftmost phrase such that the symbol immediately left of the substring is <· the leftmost symbol of the substring, and the rightmost symbol of the substring is ·> the symbol immediately to the right of the substring, and contains no other phrases except itself.

The process of reducing a sentential string using precedence relations can now be described.

If at most one precedence relation holds between any ordered pair \((A_i, A_j)\) which belongs to a grammar, the parsing algorithm proceeds as follows:

1. Enclose the input string \(Z\) with end markers denoted by @ assuming that @ <· \(Z_i\) and \(Z_i\) ·> @ for all \(Z_i \in A\).

2. For each ordered pair \(Z_i, Z_{i+1} \in A \cup \{\@\}\) insert a symbol as follows:
   
   - (a) If \(Z_i <· Z_{i+1}\) then insert a "[" so \(Z_i[Z_{i+1}\)
     - else if \(Z_i \equiv Z_{i+1}\) then no symbol is used so \(Z_iZ_{i+1}\)
     - else if \(Z_i \rightarrow Z_{i+1}\) then insert a "]" so \(Z_i]Z_{i+1}\)
     - else if no relations hold then the string is not legal.

   The new delimited string will be denoted by \(Z'\).

3. In the string \(Z'\), find the leftmost substring such that
   
   \([Z_i \ldots Z_{i+k}]\) and no "[" or "]" occur within the substring, then the substring \(Z_i \ldots Z_{i+k}\) is the handle. Reduce the handle and remove all the "[" and "]" brackets in the string \(Z'\).

   If the newly formed string is @S@, where \(S\) is the starting symbol, then the reduction is complete else go to step 2.
The following example will help to illustrate the algorithm.

**Example 2.1**

A grammar is given and its corresponding precedence relations.

**Grammar:**

\[
\begin{align*}
S & \rightarrow aBc \\
B & \rightarrow aCc \\
C & \rightarrow x
\end{align*}
\]

**Precedence Relations:**

\[
\begin{array}{c|c|c|c|c}
& B & C & a & x & c \\
\hline
B & = & & & & \\
C & & = & & \\
a & & & = & < \\
x & & & > & < \\
c & & & > &
\end{array}
\]

**Input string:** aaxcc

<table>
<thead>
<tr>
<th>Scan</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>@[a[a[x]c]c]@</td>
</tr>
<tr>
<td>2</td>
<td>@[a[aCc]c]@</td>
</tr>
<tr>
<td>3</td>
<td>@[aBc]@</td>
</tr>
<tr>
<td>4</td>
<td>@S@</td>
</tr>
</tbody>
</table>

**Operator precedence**

A significant contribution to the area of precedence grammars was given by Floyd (7). The main part of his work defines a grammar called an operator precedence grammar. The definition of operator precedence grammars however, restricts the precedence relations to pairs of terminal symbols only. Parsing a string using operator precedence relations produces a deterministic reduction, a proof of which is displayed by Fischer (5). In addition, the time bound for the recognizer is of the operator precedence grammars is that they form a relatively small subset of the context free grammars. An inclusion diagram will be developed later which shows the relationships of the various types of grammars.
Symbol precedence

A generalized approach to precedence analysis was advanced by Wirth and Weber (13). This extension to Floyd's work uses a precedence relation between adjacent symbols in a string both terminal and non-terminal. This corresponds to the intuitive example used previously where $Z_i, Z_{i+1} \in (V_N \cup V_T)$. The concepts for precedence grammars will now be stated formally since their definition will provide the basis for the extension by this thesis.

**Definition 2.1 (LEFTMOST CHARACTER DERIVATIVE)**

The symbol $A_i$ is a leftmost character derivative of $N$ if there exists a production $N \rightarrow A_i \omega$ where $N, A_i \in A$ and $\omega$ is a string over $A^*$, or the production $N \rightarrow N_1 \omega$ where $N, N_1 \in A$ and $A_i \in \text{LCD}(N_1)$.

**Definition 2.2 (RIGHTMOST CHARACTER DERIVATIVE)**

The symbol $A_j$ is a rightmost character derivative of $N$ if there exists a production $N \rightarrow \alpha A_j$ where $N, A_j \in A$ and $\alpha$ is a string over $A^*$, or the production $N \rightarrow \alpha N_1$ where $N, N_1 \in A$ and $A_j \in \text{RCD}(N_1)$.

**Definition 2.3 (PRECEDENCE GRAMMAR)**

A precedence grammar is a context free grammar for which at most one of the following relations holds between an ordered pair of symbols $(A_i, A_j)$

1. $A_i \preceq A_j$ if and only if a production $N \rightarrow \alpha A_i A_j \omega$ exists where $N, A_i, A_j \in A$ and $\alpha, \omega \in A^*$. 
An important point about precedence grammars is that they produce a deterministic parse, if and only if, no two productions contain the same right side as proved by Wirth and Weber (13).

Definition 2.4 (DETERMINISTIC PRECEDENCE GRAMMAR)

A deterministic precedence grammar is a precedence grammar whose production set contains no two productions with the same right side.

Work by Eickel, Paul, Bauer and Samelson (4) provided a set of grammars referred to as reducing transition grammars. Recently, Morris (10) has proved that the deterministic precedence grammars are a proper subset of the reducing transition grammars. The reducing transition parser has the disadvantage of producing a large number of state sets and for large grammars would be space bounded as shown by Morris (10).

Attempting to increase the subset of context-free grammars parsable by precedence techniques, Colmerauer's (2) work extended the deterministic precedence grammars to total precedence grammars as defined below.
Definition 2.5 (TOTAL PRECEDENCE GRAMMAR)

A total precedence grammar is a deterministic precedence grammar with the following change of condition 3 of definition 2.3.

(1) \( A_i \rightarrow A_j \) if and only if a production \( N \rightarrow N_i N_j \) \( \omega \) exists where \( N, N_i, N_j \in A \) and \( A_i, A_j \in A \) such that

\( A_i \in RCD(N_i) \) and \( A_j \in LCD(N_j) \) but only for \( (A_i A_j) \) pairs where the previous relation \( A_i \rightarrow A_j \) does not occur from condition 2.

The total precedence grammars represent an interesting set of grammars which include some non-deterministic grammars. Colmerauer (2) also shows that the analysis time is at most proportional to the length of the input string. In addition, the analyzer for total precedence relations is simple and similar to the intuitive example given previously.

String Precedence Systems

Attempts to find syntactical analysis techniques of greater capability led to generalizations of the previous concepts. The work with relations between adjacent symbols was extended to strings.

Bounded context grammars

Using strings and the context in which an immediate reduction occurred, Floyd (8) specified a grammar called bounded context. Floyd (8) also modifies the bounded context definition to define a grammar which is called bounded right context. Floyd (8) points out, however, that the precedence for bounded context analysis using
either definition, particularly for context strings greater than length 1, makes unreasonable demands on both time and space.

The concept of bounded right context as defined in Floyd (8) implies that one knows if a string $s_{i+1} \ldots s_{i+n}$ is a handle by examining a given finite number of characters immediately to the left of $s_{i+1}$, and in addition $s_{i+n+1} \ldots s_{i+n+k}$ is known on the right. Knuth (9) extended this notation by considering that the entire string to the left of $s_{i+1}$ is known as well as $k$ symbols to the right of $s_{i+n}$. These are known as LR(k) grammars. Terrine (12) shows that the bounded right context grammars are a proper subset of the LR(k) grammars. Thus the LR(k) grammars are a larger class of grammars than bounded right context; but as mentioned previously, the size of the parser is prohibitive for large grammars.

General precedence grammars

The previous work on precedence by Wirth and Weber (13) also included an extension to include relationships between strings. The generalized precedence grammars are termed order $(m,n)$ where $m$ and $n$ represent the length of the strings required to determine a consistent set of precedence relations. It has been proven by Cocke and Schwartz (1) that the bounded context grammars are equivalent to the precedence $(m,n)$ grammars.

McKeeman, as described by Feldman and Gries (6), also worked on an extension to Wirth and Weber's precedence grammars by using "triples" to include more information for the precedence relations. The triple corresponds, however, to the precedence $(m,n)$ grammars where the
grammar is of order (1,2) or (2,1). Work with precedence \((m,n)\)
grammars uses the same concepts as simple precedence grammars; however,
the left and right character derivatives are expanded to incorporate
more than the one symbol.

The formal definition of the \(n\)-left character derivative and
\(n\)-right character derivative \((\text{LCD}(n)\) and \(\text{RCD}(n)\)) is given and will
be later required for context resolvable grammars.

**Definition 2.6 (LEFT CHARACTER DERIVATIVE\(^{(n)}\))**

Let \(U \in V_N, Z_i \in A\) and \(u, z \in A^*\) then for some positive integers \(n\) and \(k,\)

1. A string \(z = Z_1 \ldots Z_n\) is an \(n\)-left character derivative of
   \((Uu)\) if there exists a derivation \(U \Rightarrow Z_1 \ldots Z_k,\) for \(1 \leq k \leq n\)
   and \(Z_{k+1} \ldots Z_n u' = u\) or \(Z_{k+1} \ldots Z_n\) is an \((n-k)\) left character
   derivative of \(u.\)

2. A string \(z = Z_1 \ldots Z_n\) is an \(n\)-left character derivative of \(U\)
   if there exists a derivation \(U \Rightarrow Z_1 \ldots Z_k u,\) for \(1 \leq k \leq n\) and
   \(Z_{k+1} \ldots Z_n\) is an \((n-k)\) left character derivative of \(u.\)

**Definition 2.7 (RIGHT CHARACTER DERIVATIVE\(^{(n)}\))**

Let \(U \in V_N, Z_i \in A\) and \(u, z \in A^*\) then for some positive integers \(n\) and \(k,\)

1. A string \(z = Z_n \ldots Z_1\) is an \(n\)-right character derivative of
   \((uU)\) if there exists a derivation \(U \Rightarrow Z_k \ldots Z_1,\) for \(1 \leq k \leq n\)
   and \(u'Z_n \ldots Z_{k+1} = u\) or \(Z_n \ldots Z_{k+1}\) is a \((n-k)\) right character
   derivative of \(u.\)
A string \( z = Z_n \ldots Z_1 \) is an \( n \)-right character derivative of \( U \) if there exists a derivation \( U \Rightarrow uZ_k \ldots Z_1 \) for \( 1 \leq k \leq n \) and \( Z_n \ldots Z_{k+1} \) is an \( (n-k) \)-right character derivative of \( u \).

With the use of the leftmost string sets and the rightmost string sets, just defined, the definition of 2.3 of Chapter 3 will be reformulated to define precedence relations between strings \( \alpha_i \) and \( \alpha_j \).

The left and right character derivatives are an important part of analyzing the possible string configurations and will be incorporated later as part of the extension by this thesis. The example which follows will aid in understanding these definitions.

**Example 2.2**

For a given grammar

Grammar LR: \( A \rightarrow aB, B \rightarrow bCE, C \rightarrow cD, D \rightarrow d, E \rightarrow e, F \rightarrow fGH, H \rightarrow h, J \rightarrow j, G \rightarrow g \).

The LCD\(^{(4)}\) (\( \alpha \)) set will be derived where \( \alpha = AFJ \)

\[
\text{LCD}\(^{(4)}\) (AFJ) = \{aBFJ, abCF, abcD, abed, aBfG, aBfg \}
\]

In a similar manner the RCD\(^{(3)}\) (\( \alpha \)) are derived so:

\[
\text{RCD}\(^{(3)}\) (AFJ) = \{AFj, GHj, G\alpha j, ghj, gHj \}
\]

**Definition 2.8 (PRECEDENCE (m,n) GRAMMAR)**

A precedence \((m,n)\) grammar is a context free grammar for which at most one of the following relations holds between any ordered pair of strings \( \alpha_i \) and \( \alpha_j \), where \( \alpha_i = A_m \ldots A_2 A_1 \) and \( \alpha_j = A_1 A_2 \ldots A_n \).
(1) \( a_i \equiv a_j \) if there exists a production \( N \rightarrow u_i A_{-1}\ldots A_{-n}v \)
where \( u_i A_{-m}\ldots A_{-2} = u \) or \( A_{-m}\ldots A_{-2} \in \text{RCD}^{(m-1)}(u) \)
and \( A_{-2}\ldots A_{n}v' = v \) or \( A_{-2}\ldots A_{n} \in \text{LCD}^{(n-1)}(v) \)

(2) \( a_i < a_j \) if there exists a production \( N \rightarrow u_i A_{-1}\ldots A_{-n}v \)
where \( u_i A_{-m}\ldots A_{-2} = u \) or \( A_{-m}\ldots A_{-2} \in \text{RCD}^{(m-1)}(u) \)
and \( A_{1}\ldots A_{n}v' = v \) or \( A_{1}\ldots A_{n} \in \text{LCD}^{(n)}(v) \)

(3) \( a_i \Rightarrow a_j \) if \((a)\) there exists a production \( N \rightarrow u_i N_{1}\ldots A_{-n}v \)
where \( A_{-m}\ldots A_{-1} \in \text{RCD}^{(m)}(u_i N_1) \)
and \( A_{2}\ldots A_{n}v' = v \) or \( A_{2}\ldots A_{n} \in \text{LCD}^{(n-1)}(v) \)
\( \) or \((b)\) there exists a production \( N \rightarrow u_i N_{1}\ldots N_{j}v \)
where \( A_{-m}\ldots A_{-1} \in \text{RCD}^{(m)}(u_i N_1) \)
and \( A_{1}\ldots A_{n} \in \text{LCD}^{(n)}(N_j v) \)

A grammar is said to be a precedence grammar of order \((r,s)\), if for a
given precedence \((m,n)\) grammar of definition 2.8

(1) \( r = \text{maximum} \ |a_j| \)
and (2) \( s = \text{maximum} \ |a_j| \).
Example 2.3

Grammar: 1. $A \rightarrow A; s$

Sets: $U$ LCD$^{(1)}$(U) $A$ B, d

2. $A \rightarrow B$

$B, d, s$

3. $B \rightarrow d; B$

$B, d$

4. $B \rightarrow d$

<table>
<thead>
<tr>
<th></th>
<th>LCD$^{(1)}$(U)</th>
<th>RCD$^{(1)}$(U)</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>A, B, d</td>
<td>B, d, s</td>
</tr>
</tbody>
</table>

(1) Precedence (1.1) matrix:

A B d s ;

A ≈

B ≻

d ≻≈

s ≻

; ≡ ≪ ≡

The precedence matrix shows that $d ≻;$ and $d ≡;$ which means the pair $(d,;)$ is not of order (1.1). They may however possess a greater order.

(2) The precedence relations for the pair $(d,;)$ will now be derived for a (1,2) order. The production $B \rightarrow d; B$ yields the precedence relations: $d ≡ ;B$ and $d ≡ ;d$. The production $A \rightarrow A; s$ yields the relation: $d ≻ ;s$. Since the three are unique the grammar is a precedence (1,2) grammar.
CONTEXT RESOLVABLE GRAMMARS

Precedence grammars presented thus far have properties which make them attractive candidates for syntax directed compilers. It would be desirable to define as large a set of grammars as possible without sacrificing the efficiency of the analyzer. The precedence \((m,n)\) grammars provide a larger set of precedence grammars than just simple precedence, however only a very general algorithm has been described which would determine whether or not a grammar is precedence of order \((m,n)\). Also, no analyzer is specified for parsing precedence \((m,n)\) grammars. It would be advantageous, however to extend the precedence \((m,n)\) grammars even farther. In particular, it would be advantageous to remove the left-to-right scan implied by precedence \((m,n)\) grammars. Removing this implication we effectively get an extension of Colmerauer's (2) total precedence grammars. The extended grammars will be called context resolvable grammars; and the necessary table structure and parsing algorithm will be defined for context resolvable grammars. An informal explanation will be presented first, followed by symbolic definitions and examples.

Context resolvable grammars are essentially grammars in which any precedence conflict between two adjacent symbols in a string, being parsed, can be resolved by looking at more symbols. In particular it may be required to go to the left of the pair for this extra context; it may be required to look to the right for additional context, or both. In any case, the conflict can be resolved by including at most \(m\) characters on the left and at most \(n\) characters to
the right where \( m \) and \( n \) are fixed integers for the grammar. If for a grammar, it is not possible to fix these numbers \( m \) and \( n \), then the grammar is not context resolvable.

The crucial factor in resolving a precedence conflict between two symbols say \( A_i \) and \( A_j \), is that it must be possible to define the necessary context (some strings) which resolves the conflict for any occurrence of \( A_i \) and \( A_j \) in a sentential form at any step in the parse. These necessary contextual strings can be divided into two groups, namely the context string (if any) to the left of \( A_j \) to resolve the conflict and the context string (if any) needed to the right of \( A_i \) to resolve the conflict. Since in general the string \( A_i A_j \) may appear in many different situations, there will generally exist many different left contextual strings, any one of which can resolve the conflict. These left contextual strings comprise a set called the left contextual set. Similarly, any one of a number of right contextual strings may exist to resolve the conflict. These right contextual strings will comprise a set called the right contextual set.

Before defining these sets more formally consider some simple examples.

**Example 3.1**

For the grammar

\[
A \rightarrow abcd
\]

\[
A \rightarrow aBce
\]

\[
B \rightarrow b
\]

in this grammar a precedence conflict exists between \( b \) and \( c \) because \( b \prec c \) holds and \( b \rightarrow c \) holds. Also, \( a \prec b \) and \( a \prec b \) holds.

(1) The conflict between the \( a \) and \( b \) can be resolved by looking at the symbols to the right of \( a \). The productions show
that if 'a' is followed by 'bcd' then the precedence relation \( a \equiv b \) holds. If the string 'bee' is found then the precedence relation \( a < b \) holds.

(2) The conflict between the 'b' and 'c' can be resolved by including two more symbol to the right. If 'b' is followed by a 'cd' then the relation \( b = c \) holds; else, 'b' followed by a 'ce' requires that the relation \( b > c \) hold.

In neither case does it help to look to the left to resolve the conflict. Thus the left contextual set for 'ab' is empty and the right contextual sets contain the strings 'bcd' for \( \equiv \) and 'bee' for \( < \). Also, the left contextual set for 'bc' is empty, while the right contextual set contains a 'cd' for \( \equiv \) and a 'ce' for \( > \).

Example 3.2

For the grammar

\[
A \rightarrow \text{abcd} \\
A \rightarrow \text{ab} \text{cd} \\
B \rightarrow b
\]

the same precedence conflicts exist as in the previous example. This time however the relation \( a \equiv b \) holds within the same context that \( a < b \) holds. Similarly the relation \( b = c \) and \( b > c \) holds within the same context. Therefore these conflicts cannot be resolved by looking at more context to the left or right since the only available context to the left and right is the same for both relations.

Example 3.3

Grammar:

\[
A \rightarrow \text{abcd} \\
A \rightarrow \text{ab} \text{Cc} \\
B \rightarrow b \\
C \rightarrow c \\
C \rightarrow \text{Cc}
\]
The conflicts can be resolved by looking to the right but there is no way to put a bound on how far we have to go. The legal strings of the grammar are $abc^n d$ and $abc^n e$ for a positive integer $n$. Ultimately in a legal string we resolve the conflict by detecting a string of the form $bc^n d$ or $bc^n e$ to the right of 'a'. But since any number of 'c's can occur between the 'b' and 'd' or the 'b' and 'e', it is not possible to guarantee a finite amount of context to resolve the conflict.

Now we can proceed to give a more formal definition of the left and right contextual sets. Note that in defining a left (right) contextual set for a conflict pair $(A_i, A_j)$, the set will be partitioned into three subsets one for each precedence relation and in addition each relational subclass will be subdivided into subsets — one for each production which generates that relation. Thus it will be convenient to talk about the left (right) contextual sets of the $(A_i, A_j)$ pair for which $A_i \equiv A_j$ holds, the left (right) contextual sets for which $A_i < A_j$ holds, and finally the left (right) contextual sets for which $A_i > A_j$ holds. The left contextual sets will be denoted by $L^k_{p,n}(A_i, A_j)$ where $p$ represents the precedence relation, $n$ is the number of the production (assuming some ordering of the productions) which generates that set for the respective relation, and $k$ is the maximum length of the strings in the set. Similarly the right contextual sets will be denoted $R^k_{p,n}(A_i, A_j)$.

**Definition 3.1 (LEFT CONTEXTUAL SETS)**

The three $L^k_{p,n}(A_i, A_j)$ sets are defined as follows, where

$$
\alpha_i = A_{-k} \cdots A_{-2} A_{-1}
$$
(1) \( \alpha_i \in L^k_{\geq,n}(A_{-1},A_1) \) if there exists a production of the form
\[ N \rightarrow uA_{-1}A_1v \text{ where } u'A_{-k}A_2 = u \text{ or } A_{-k}A_2 \in \text{RCD}(k-1)(u) \]

(2) \( \alpha_i \in L^k_{<,n}(A_{-1},A_1) \) if there exists a production of the form
\[ N \rightarrow uA_{-1}N_1v, \text{ where } A_1 \in \text{LCD}(1)(N_1) \text{, where } u'A_{-k}A_2 = u \text{ or } A_{-k}A_2 \in \text{RCD}(k-1)(u) \]

(3) \( \alpha_i \in L^k_{>,n}(A_{-1},A_1) \) if there exists a production of the form
\[ N \rightarrow uN_1N_1v \text{ where } A_1 \in \text{LCD}(1)(N_1) \]
or
\[ N \rightarrow uN_1A_1v \text{ where } A_{-k}A_1 \in \text{RCD}(k)(uN_1) \]

**Definition 3.2 (RIGHT CONTEXTUAL SETS)**

The three \( R^k_{p,n} \) sets are defined as follows, where \( \alpha_j = A_1A_2\ldots A_k \)

(1) \( \alpha_j \in R^k_{\geq,n}(A_{-1},A_1) \) if there exists a production of the form
\[ N \rightarrow uA_{-1}A_1v \text{ where } A_2A_kv' = v \text{ or } A_2A_k \in \text{LCD}(k-1)(v) \]

(2) \( \alpha_j \in R^k_{<,n}(A_{-1},A_1) \) if there exists a production of the form
\[ N \rightarrow uA_{-1}N_1v \text{ where } A_1A_k \in \text{LCD}(k)(N_1v) \]

(3) \( \alpha_j \in R^k_{>,n}(A_{-1},A_1) \) if there exists a production of the form
\[ N \rightarrow uN_1A_1v \text{ where } A_1A_k \in \text{RCD}(k)(uN_1) \]
and
\[ A_2A_kv' = v \text{ or } A_2A_k \in \text{LCD}(k-1)(v) \]
or there exists a production of the form
\[ N \rightarrow uN_1N_1v \text{ where } A_1A_k \in \text{LCD}(k)(N_1v) \]
The following examples will help illustrate the concepts described above.

**Example 3.4**

Grammar: 1. \( S \rightarrow axYB \)
2. \( S \rightarrow bxyB \)
3. \( Y \rightarrow y \)
4. \( B \rightarrow b \)

1. By definition 2.8 a conflict for the pair \((x,y)\) exists since \(x < y\) and \(x \neq y\).
2. Left contextual sets:
   \( L_{c1}^{2}(x,y) = \{ax\} \)
   \( L_{c2}^{2}(x,y) = \{bx\} \)

The left contextual sets are unique for each relation and hence the conflict can be resolved such that \(ax < y\) and \(bx \neq y\) holds. The pair \((x,y)\) is thus said to be of order \((2,1)\).

**Example 3.5**

Grammar: 1. \( S \rightarrow axYzc \)
2. \( S \rightarrow axyzd \)
3. \( S \rightarrow axyze \)
4. \( Y \rightarrow y \)

1. Using definition 2.8 the conflict pair \((x,y)\) exists where \(x \neq y\) and \(x < y\).
2. Right contextual sets:
   \( R_{c1}^{3}(x,y) = \{vzc\} \)
   \( R_{c2}^{3}(x,y) = \{yzd\} \)
   \( R_{c3}^{3}(x,y) = \{yze\} \)
The right contextual sets are unique for each relation and the conflict can be resolved such that \( x \preceq yzc \) or \( x \preceq yzd \) or \( x \preceq yze \) holds. The pair \((x,y)\) is said to be of order \((1,3)\).

(3) The conflict pair \((y,z)\) also exists were \( y \preceq z \) and \( y \succ z \)

(4) The right contextual sets:
\[
R_{>1}^2 (y,z) = \{zc\}, \quad R_{>2}^2 (y,z) = \{zd\}, \quad R_{>3}^2 (y,z) = \{ze\}
\]
and since the sets are unique the conflict can be resolved where \( y \succ zc \), \( y \preceq zd \), and \( y \preceq ze \). The pair \((y,z)\) is therefore of order \((1,2)\).

**Definition 3.3 (CONTEXT RESOLVABLE GRAMMAR)**

A context resolvable grammar is a precedence \((m,n)\) grammar with the following restriction of the second part of condition 3 of definition 2.8.

\( (3') \) except where \( \alpha_i < \alpha_j \) from condition 2.

The definition for context resolvable grammars is structured in a way such that all symbol pairs \((A_i,A_j)\) which are at most of order \((1,1)\) total precedence, can be partitioned out first. Those pairs which are of greater order can then be scanned for context to the left and right to determine a minimum order \((m,n)\) which is less than some finite order \((k,k)\). The left context (if any) is obtained using the left contextual sets of definition 3.1 and context (if any) to the right of the conflict pair is obtained using Definition 3.2 for the right contextual sets. The conditions for these sets to determine a precedence relation between the strings \( \alpha_i \) and \( \alpha_j \) according to Definition 3.3 will be given
formally in the generating algorithm, later. It should be noted that in Example 3.5 if the left contextual sets had been derived for the pair \((x,y)\) with \(k = 2\), then each set would have been identically, \(\{ax\}\). Attempting to derive more context on the left by setting \(k = 3\) causes the generation of a null set since no additional context exists or can be derived per Definition 3.1 (2). Any value of \(k\), greater than a \(k\) producing a null set, also produces a null set. Thus if a conflict pair has not been resolved by context say on the left for a \(k = 2\) and setting \(k = 3\) produces a null set, then the pair cannot be resolved for any value of \(k\) on the left. Similarly, if a right contextual set is null, then the conflict cannot be resolved on the right. Finally, if both the left and right contextual sets contain a null set, then the conflict is not context resolvable for any value of \(k\). For those pairs which are precedence of order \((m,n)\) a table can be constructed which contains the resolving left and/or right strings and the corresponding precedence relation.

This form of partitioning allows for utilization of the standard precedence analyzer with only slight modification to do a "table lookup" to resolve any pair \((A_i,A_j)\) which is not of order \((1,1)\) total precedence. Prior to displaying an example, consideration will be given for construction of the precedence matrix. The precedence relations \(<*, \neq, \succ>\), are determined for each ordered pair \((A_i,A_j)\) as specified by the context resolvable grammar Definition 3.3, where \(m = 1\) and \(n = 1\), and assigned to the \((i,j)\) position of a matrix \(M\). Those pairs \((A_i,A_j)\)
for which no precedence relation exists will have \( M(i,j) \) empty. If more than one precedence relation holds for an ordered pair \((A_i, A_j)\) then a '?' is placed in \( M(i,j) \).

The following example will show a context resolvable grammar using a precedence matrix to represent the relations derived.

**Example 3.3**

<table>
<thead>
<tr>
<th>Grammar:</th>
<th>Matrix:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ( S \rightarrow axZc )</td>
<td>( S Y Z a b x y z c d )</td>
</tr>
<tr>
<td>2. ( S \rightarrow bxYd )</td>
<td>( S )</td>
</tr>
<tr>
<td>3. ( S \rightarrow axyc )</td>
<td>( Y )</td>
</tr>
<tr>
<td>4. ( S \rightarrow axzd )</td>
<td>( Z )</td>
</tr>
<tr>
<td>5. ( Y \rightarrow y )</td>
<td>( a )</td>
</tr>
<tr>
<td>6. ( Z \rightarrow z )</td>
<td>( b )</td>
</tr>
</tbody>
</table>

(1) The precedence matrix shows a conflict for the pair of symbols \((x, y)\) where \( x \neq y \) and \( x < y \).

Since \( L_{\neq, 3}^2 (x, y) = \{ax\} \) and \( L_{\neq, 2}^2 (x, y) = \{bx\} \)

The \((x, y)\) pair is of order \((2,1)\) where \( ax \neq y \) and \( bx < y \).

(2) Another conflict pair of the matrix is \((x, z)\) where \( x \neq z \) and \( x < z \).

Since \( R_{\neq, 4}^2 (x, z) = \{zd\} \) and \( R_{\neq, 1}^2 (x, z) = \{zc\} \)

The pair \((x, z)\) is of order \((1,2)\) where \( x \neq zd \) and \( x < z \).

(3) The grammar as a whole then is context resolvable of at most order \((2,2)\).
Set Relations

The results of this chapter using an extension to the definition of precedence \((m,n)\) grammars provides a larger subset of the context free grammars which can be parsed proportional to the length of the input string. The results of grammar classifications by set inclusion relations will now be derived to show the relationship between context resolvable grammars and the LR\((k)\) grammars which Knuth (9) proves defines exactly those languages which are deterministic.

Lemma 1. The results of Morris (10) show that the grammar \(G_1:\)

\[
S \rightarrow E, \ S \rightarrow bB, \ A \rightarrow G_2V, \ A \rightarrow WV, \ C \rightarrow PC, \ C \rightarrow P \\
B \rightarrow FQ_3, \ B \rightarrow D, \ D \rightarrow XD, \ D \rightarrow X, \ P \rightarrow Q, \ Q \rightarrow 2, \ X \rightarrow V, \\
V \rightarrow 3, \ W \rightarrow Y, \ Y \rightarrow 0, \ E \rightarrow aA, \ G_2 \rightarrow WA, F \rightarrow YB
\]

which defines the language \(L_1 = \{a^n_1b^n_2 | (m,n) \} \) is deterministic.

Lemma 2. The grammar \(G_1\), of Lemma 1, is not a context resolvable grammar.

Proof:

(1) The precedence relations derived by definition 2.5 show a conflict for the pair \((Y,V)\) such that \(Y < V\) and \(Y > V\) from the productions \(F \rightarrow YB\) and \(A \rightarrow WV\) respectively.

(2) The left contextual sets:

\[
L^{<,1}_{Y,V} = \{\}, \ L^{>,2}_{Y,V} = \{\}
\]

Hence not resolvable on the left for any \(k\).
(3) The right contextual sets:
\[ R^2_{<*,1}(Y,V) = \{VD, VX, VV, VL\}, R^2_{>,2}(Y,V) = \{\emptyset\} \]
Since the \( R^2_{>,2}(Y,V) \) set is null the conflict is not resolvable on the right for any \( k \). The grammar is therefore not context resolvable.

**Theorem 1.** The set of LR(k) grammars is not a proper subset of the context resolvable grammars.

**Proof:** By Lemma 1 and Lemma 2.

**Lemma 3.** The grammar \( G_3 \cup G_4 \):
\[
S \rightarrow X, X \rightarrow aN, X \rightarrow aX, N \rightarrow bA, N \rightarrow bNA,
\]
\[
A \rightarrow a, S \rightarrow VZ, Z \rightarrow ac, Z \rightarrow aZ, V \rightarrow aB, V \rightarrow aVB, B \rightarrow b,
\]
\[
S \rightarrow F, F \rightarrow Hd, F \rightarrow Fd, H \rightarrow De, H \rightarrow DHe, D \rightarrow d,
\]
\[
S \rightarrowYW, Y \rightarrow fd, Y \rightarrow Yd, W \rightarrow Ed, W \rightarrow EWd, E \rightarrow e
\]
which defines the language \( L_3 \cup L_4 \) is not an LR(k) grammar.

**Proof:** Colmerauer (2) proves that the language generated by the union of the grammar \( G_3 \), which defines the language \( L_3 = \{a^i b^j a^j \cup a^i b^j a^j c\} \) \( i,j > 0 \) and \( G_4 \) which defines the language \( L_4 = \{fd^i e^j d^j \cup d^j e^j d^j\} \) \( for i,j > 0 \) is a non-deterministic language \( (L_3 \cup L_4) \).
Lemma 4. The grammar $G_3 \cup G_4$ is a context resolvable grammar.

Proof:

(1) The left and right character derivatives:

<table>
<thead>
<tr>
<th>P</th>
<th>LCD$^{(1)}$(P)</th>
<th>RCD$^{(1)}$(P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>X,a</td>
<td>X,N,A,a,Z,c</td>
</tr>
<tr>
<td>X</td>
<td>a</td>
<td>X,N,A,a</td>
</tr>
<tr>
<td>N</td>
<td>b</td>
<td>A,a</td>
</tr>
<tr>
<td>V</td>
<td>a</td>
<td>B,b</td>
</tr>
<tr>
<td>B</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>Z</td>
<td>a</td>
<td>Z,c</td>
</tr>
<tr>
<td>A</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>S</td>
<td>F,H,D,d</td>
<td>F,d</td>
</tr>
<tr>
<td>F</td>
<td>H,D,d,F</td>
<td>d</td>
</tr>
<tr>
<td>H</td>
<td>D,d</td>
<td>e</td>
</tr>
<tr>
<td>Y</td>
<td>f,Y</td>
<td>d</td>
</tr>
<tr>
<td>W</td>
<td>E,e</td>
<td>d</td>
</tr>
<tr>
<td>D</td>
<td>d</td>
<td>d</td>
</tr>
<tr>
<td>E</td>
<td>e</td>
<td>e</td>
</tr>
</tbody>
</table>

(2) The precedence matrix generated by Definition 3.3 where $m = 1$ and $n = 1$. 
The set of context resolvable grammars is not a proper subset of the LR(k) grammars.

Proof: By Lemma 3 and Lemma 4.
The following inclusion diagram will show the previously known set relations and include the results of Theorem 1 and Theorem 2. Example 3.3 also shows a context resolvable grammar which is not a total precedence grammar.

![Diagram showing grammar relations]

Figure 3.1. Grammar relations
ALGORITHMS AND EXAMPLES

Generating Algorithm and Analyzer

In this chapter the necessary algorithms to determine if a given grammar is context resolvable will be presented. The process also involves the building of the table which contains the left and right context for resolving the conflict if possible. The information obtained from these two algorithms can then be used by the context resolvable analyzer to parse a string generated by the grammar.

First we will define a simple table structure which will contain the left and right sets, for a given conflict pair \((A_i, A_j)\), which contains the corresponding precedence relation for each element of the left or right set. The table entry also provides an indicator (denoted by +) which specifies the next table entry is an alternative set or, (denoted by \(\#\)) when no more alternative entries exist.

Definition 4.1 (CONTEXT RESOLUTION TABLE)

A context resolution table is denoted by \(\text{Resolve} (A_i, A_j) = (\alpha_i, \alpha_j, \rho, \eta)\)

1. \(A_i\) and \(A_j\) are the symbols in conflict
2. \(\alpha_i\) and \(\alpha_j\) are the resolving strings
3. \(\rho\) is the precedence relation \(\prec, =\) or \(\succ\) which corresponds to that context
4. \(\eta\) is the indicator that an alternative entry, the next entry in the table, exists for the conflict pair.

This allows the parser of a context resolvable grammar to first determine if a precedence relation exists for a pair \((A_i, A_j)\) or switch to a table
reference for resolving pairs which show a conflict of precedence.

Using the previous definitions, an algorithm for generating the necessary matrix and resolving table for a specified grammar and some finite positive integer k will be given. A grammar which does not satisfy the conditions of the definitions is not context resolvable.

Test algorithm

The algorithm to test a grammar requires that the precedence matrix be generated for all pairs $A_i$ and $A_j$ which belong to the vocabulary. The matrix is then searched for precedence conflicts. Each conflict pair is then examined by obtaining each production which causes a precedence relation to be generated for that pair. The left contextual sets for the pair $(A_i, A_j)$ are generated setting $k = 2$. The union of each set with the same relation is obtained and then the intersection of the sets for each relation is performed. If the sets are disjoint, then the conflict pair $(A_i, A_j)$ is resolvable on the left for a string of length 2. If they are not disjoint, then a similar process is performed for the right contextual sets.

If the left (or right) sets are disjoint the context is resolvable on the left (or right), and so each member of the left (or right) sets is placed in an entry of the resolve table for the $(A_i, A_j)$ pair with its corresponding precedence relation. If the right sets and left sets are not disjoint, the pair may still be resolvable, if for any two left (right) sets which are not disjoint, their corresponding right (left) sets are disjoint then the conflict can be resolved. In this case, for each unique left or right set, these elements determine the corresponding precedence relation. The remaining relational sets must then be used pair wise (both
left and right) to determine the precedence relation.

Consider the following example:

**Example 4.1**

Given the contextual sets:

\[ L_{*,1}^2(x,y) = \{ax\}, \quad L_{*,2}^2(x,y) = \{bx\}, \quad L_{*,3}^2(x,y) = \{ax\} \]

and

\[ R_{*,1}^2(x,y) = \{yc\}, \quad R_{*,2}^2(x,y) = \{yd\}, \quad R_{*,3}^2(x,y) = \{yd\} \]

(1) Since \( \{ax, bx\} \cap \{ax\} \neq \emptyset \) the pair is not left resolvable.

(2) Since \( \{yc, yd\} \cap \{yd\} \neq \emptyset \) the pair is not right resolvable.

(3) However \( L_{*,1}^2(x,y) \cap L_{*,3}^2(x,y) \neq \emptyset \) but \( R_{*,1}^2(x,y) \cap R_{*,3}^2(x,y) = \emptyset \) so the conflict pair \( (x,y) \) is resolvable.

(4) The set \( L_{*,2}^2(x,y) = \{bx\} \) is unique so let

\[ \text{Resolve } (x,y) = (bx, \phi, <\cdot, +) \]

And \( R_{*,1}^2(x,y) = \{yc\} \) is unique so let

\[ \text{Resolve } (x,y) = (\phi, yc, <\cdot, +) \]

The remaining sets are \( L_{*,3}^2(x,y) = \{ax\} \) and \( R_{*,3}^2(x,y) = \{yd\} \)

so Resolve \( (x,y) = (ax, yd, \neq, \phi) \).

If the left and right sets had not been pairwise disjoint then the grammar would not have been resolvable for \( k = 2 \), or the grammar was not of order \((1,2)\) or \((2,1)\) or \((2,2)\). The value of \( k \) could be incremented to 3 and the process repeated. The iteration can continue until some value of \( k \) which is specified or if for a value of \( k \) a left (or right) contextual set becomes null, then no more context is available for the left (or right) sets so they can no longer be used to determine resolvability. If both left and right sets contain null sets then the conflict is not resolvable.
Definition 4.2 (CONTEXT RESOLVABLE GENERATOR)

For a context free grammar specified by a set of productions (some ordering is given) and a given finite positive integer $k$, the following steps must be performed in sequence.

(1) Using Definition 3.3 generate the precedence matrix for all $(A_i, A_j)$ pairs with $m = 1$ and $n = 1$.

(2) For each $(A_i, A_j)$ ordered pair in the matrix $M$ which contains a conflict $(A_i ? A_j)$ perform the following sequence.

2.1 Find each production which generates a precedence relation for the pair $(A_i, A_j)$.

2.2.0 Set $k = 2$.

2.2.1 For each of the productions generate the $L_{p,n}^k (A_i, A_j)$ sets.

2.2.2 If $\bigcup_{n} L_{<,n}^k \bigcap \bigcup_{n} L_{=,n}^k \bigcap \bigcup_{n} L_{>,n}^k = \phi$ then the $(A_i, A_j)$ conflict pair is context resolvable on the left by strings of length $k$. Construct the related part of the Resolve table using Definition 4.1.

2.2.3 If the conflict pair was not left resolvable then generate the $R_{p,n}^k (A_i, A_j)$ sets.

2.2.4 If $\bigcup_{n} R_{<,n}^k \bigcap \bigcup_{n} R_{=,n}^k \bigcap \bigcup_{n} R_{>,n}^k = \phi$ then the $(A_i, A_j)$ conflict pair is context resolvable on the right by strings of length at most $k$. Construct the related part of Resolve table using Definition 4.1.

2.2.5 If $\bigcap_{p} \left( \bigcup_{n} R_{p,n}^k (A_i, A_j) \right) \neq \phi$ and $\bigcap_{p} \left( \bigcup_{n} L_{p,n}^k (A_i, A_j) \right) \neq \phi$ then the ordered pair $(A_i, A_j)$ is resolvable by left and
right context only if for each \( R^k_{p,i} \cap R^k_{\lambda,j} \neq \emptyset \),
then \( L^k_{p,i} \cap L^k_{\lambda,j} = \emptyset \) or for each \( L^k_{p,i} \cap L^k_{\lambda,j} \neq \emptyset \) then
\( R^k_{p,i} \cap R^k_{\lambda,j} = \emptyset \) where \( p, \lambda = <*, \cdot, \cdot> \) and \( p \neq \lambda \).

Construct the related parts of the Resolve table.

2.2.6 If the pair \((A_i, A_j)\) was not resolvable set \( k = k + 1 \)
and if \( k > \omega \) then go to 2.2.7 else go to step 2.2.1.

2.2.7 If all \((A_i, A_j)\) pairs have been resolved then
successful completion.

The generator algorithm provides a general description of the data
required for the parsing algorithm since specific details would be implementa­tion dependent. The general structure presented is meant to convey
the necessary concepts to set up proper data tables.

Analyzer

The analyzer for a string generated by a context resolvable grammar
is similar to the informal discussion in Chapter 2. The process however
requires more details for a practical implementation. The basic analyzer
requires two stacks which will be denoted \( S(i) \) and \( I(k) \). The algorithm
uses a procedure \textsc{PUSH}\_\textsc{S}(X) which places the value of \( X \) at the top of the
stack \( S \) and \textsc{PULL}\_\textsc{S} which is a procedure that returns the value of the
top element of \( S \) and deletes it from stack \( S \). Corresponding procedures
are used for the stack \( I \). The procedure \textsc{MATRIX}(S,I) finds the precedence
relation between the top element of stack \( S \) and the top element of the
stack \( I \). If \textsc{MATRIX} does not find a precedence relation for the pair it
signifies an error. If the precedence relation found by \textsc{MATRIX} is a
conflict pair, the procedure uses \textsc{RESOLVE} to determine the precedence
relations. If RESOLVE cannot find the proper left context on the S stack or the necessary right context on the I stack, then an error condition exists. Upon locating a substring which can be reduced, the analyzer uses REDUCE which searches the production set for a production whose right side corresponds to the string indicated by the top elements of the I stack and upon finding the production replaces the handle on top of the I stack by the left part of the production. If REDUCE cannot find a production whose right side is identical to the string indicated on the top of the I stack it signifies an error.

The analyzer used the two stacks by starting with the input string to be analyzed, contained in the I stack and appended with an end marker '@'. The S stack contains only an end marker '@'. The assumption is that '@' '(' A1 and A1 ')' '@' holds. The process proceeds by taking the top element of the I stack and places it onto the S stack. This is repeated until the top of the S stack is ')' in precedence than the top of the I stack. The process then proceeds to place the elements from the S stack onto the I stack until a '<' precedence is found between the top elements of S and I respectively. The handle is then the top of the I stack and is reduced. The process is then repeated. The iteration continues until an error is found or the top of the S stack contain a '@' and the elements of the stack I are 'S@', which indicates a successful parse.
The formal definition of the algorithm will be given in a pseudo-
ALGOL format. An example parse follows the definition which will aid
in understanding the procedure.

**Definition 4.3 (CONTEXT RESOLVABLE ANALYZER)**

**PROCEDURE**: PUSH\_S(X); \(i = i+1\); \(S(i) = X\); END;

**PROCEDURE**: PULL\_S; PULL\_S = S(i); \(i = i-1\); END;

**PROCEDURE**: PUSH\_I(X); \(k = k-1\); \(I(k) = X\); END;

**PROCEDURE**: PULL\_I; PULL\_I = I(k); \(k = k+1\); END;

**PROCEDURE**: RESOLVE (S(i), I(k));

- IF NOT RESOLVABLE THEN ERROR (S(i), I(k))
- ELSE PRECEDENCE = ('<', v, '=' v, '>', v')'; END;

**PROCEDURE**: MATRIX (S(i), I(k));

- IF M(S(i), I(k)) = \(\phi\) THEN ERROR (S(i), I(k))
- ELSE PRECEDENCE = M(S(i), I(k));
- IF PRECEDENCE = '?' THEN RESOLVE (S(i), I(k)); END;

**PROCEDURE**: REDUCE (I(k) ..... I(#k));

- IF I(k) ..... I(#k) \(\notin\) RIGHT PART (PRODUCTION)
  THEN ERROR (I(k) ..... I(#k))
- ELSE; BEGIN; I(#k) = LEFT PART (PRODUCTION);
  \(k = #k\); END;

**END**;

**START**: \(S(0) = '@'; i = 0; k = 1; I(\text{LENGTH}(I) + 1) = '@'\);

**SCAN**: PRECEDENCE = MATRIX (S(i), I(k));

- IF PRECEDENCE = \(\Rightarrow\) THEN BEGIN; \#k = k - 1;
SEARCH:
BEGIN; PUSH_I (PULL_S);
PRECEDENCE = MATRIX (S(i), I(k));
IF PRECEDENCE ≠ ⇑ THEN GO TO SEARCH;
ELSE; BEGIN; REDUCE (I(k)...I(#k));
IF S(i) = '@' I(k + 1) = '@' & I(k) = STARTING SYMBOL THEN GO TO SUCCESS;
END; END;
ELSE; PUSH_S (PULL_I);
GO TO SCAN;
SUCCESS: STOP;

Example 4.2

Grammar: S → axZc, S → bxYd, S → axyd, Y → y, Z → Y.
Precedence Relations: y ▷ c, b ≡ x, a ≡ x, Z ≡ c, Y ▷ c, Y ≡ d, x ≡ Z

Resolution Table:

<table>
<thead>
<tr>
<th>RESOLVE</th>
<th>α_i</th>
<th>α_j</th>
<th>ρ</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>x,y</td>
<td>ax</td>
<td>φ</td>
<td>&lt;</td>
<td>+</td>
</tr>
<tr>
<td>x,y</td>
<td>bx</td>
<td>φ</td>
<td>≡</td>
<td>φ</td>
</tr>
<tr>
<td>y,d</td>
<td>bxy</td>
<td>φ</td>
<td>▷</td>
<td>+</td>
</tr>
<tr>
<td>y,d</td>
<td>axy</td>
<td>φ</td>
<td>≡</td>
<td>φ</td>
</tr>
<tr>
<td>x,y</td>
<td>φ</td>
<td>yc</td>
<td>&lt;</td>
<td>+</td>
</tr>
<tr>
<td>x,y</td>
<td>bx</td>
<td>φ</td>
<td>&lt;</td>
<td>+</td>
</tr>
<tr>
<td>x,y</td>
<td>ax</td>
<td>yd</td>
<td>≡</td>
<td>φ</td>
</tr>
</tbody>
</table>
An underscore will mark the right end of the I stack for > precedence.

<table>
<thead>
<tr>
<th>S-stack</th>
<th>ρ</th>
<th>I-stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>@</td>
<td>&lt;•</td>
<td>b x y d @</td>
</tr>
<tr>
<td>@ b</td>
<td>↓</td>
<td>x y d @</td>
</tr>
<tr>
<td>@ b x</td>
<td>&lt;•</td>
<td>y d @</td>
</tr>
<tr>
<td>@ b x y</td>
<td>→</td>
<td>d @</td>
</tr>
<tr>
<td>@ b x</td>
<td>&lt;•</td>
<td>y d @</td>
</tr>
<tr>
<td>@ b x Y</td>
<td>↓</td>
<td>Y d @</td>
</tr>
<tr>
<td>@ b x Y d</td>
<td>→</td>
<td>@</td>
</tr>
<tr>
<td>@ b x Y</td>
<td>↓</td>
<td>d @</td>
</tr>
<tr>
<td>@ b x</td>
<td>↓</td>
<td>Y d @</td>
</tr>
<tr>
<td>@ b</td>
<td>↓</td>
<td>x y d @</td>
</tr>
<tr>
<td>@</td>
<td>&lt;•</td>
<td>b x y d @</td>
</tr>
<tr>
<td>@</td>
<td>&lt;•</td>
<td>S @</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S-stack</th>
<th>ρ</th>
<th>I-stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>@</td>
<td>&lt;•</td>
<td>a x y c @</td>
</tr>
<tr>
<td>@ a</td>
<td>↓</td>
<td>x y c @</td>
</tr>
<tr>
<td>@ a x</td>
<td>&lt;•</td>
<td>y c @</td>
</tr>
<tr>
<td>@ a x y</td>
<td>→</td>
<td>c @</td>
</tr>
<tr>
<td>@ a x</td>
<td>&lt;•</td>
<td>y c @</td>
</tr>
<tr>
<td>@ a x Z</td>
<td>↓</td>
<td>Y c @</td>
</tr>
<tr>
<td>@ a x Z</td>
<td>↓</td>
<td>c @</td>
</tr>
<tr>
<td>@ a x Z c</td>
<td>→</td>
<td>@</td>
</tr>
<tr>
<td>@</td>
<td>&lt;•</td>
<td>b x y d @</td>
</tr>
<tr>
<td>@</td>
<td>&lt;•</td>
<td>a x Z c @</td>
</tr>
<tr>
<td>@</td>
<td>&lt;•</td>
<td>S @</td>
</tr>
</tbody>
</table>
(3) Input string = bxyc

<table>
<thead>
<tr>
<th>I-stack</th>
<th>$\rho$</th>
<th>I-stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>@</td>
<td>•</td>
<td>b x y c @</td>
</tr>
<tr>
<td>@ b</td>
<td>=</td>
<td>x y c @</td>
</tr>
<tr>
<td>@ b x</td>
<td>•</td>
<td>y c @</td>
</tr>
<tr>
<td>@ b x y</td>
<td>••</td>
<td>c @</td>
</tr>
<tr>
<td>@ b x</td>
<td>=</td>
<td>y c @</td>
</tr>
<tr>
<td>@ b x Y</td>
<td>••</td>
<td>c @</td>
</tr>
<tr>
<td>@ b x</td>
<td>=</td>
<td>y c @</td>
</tr>
<tr>
<td>@ b</td>
<td>=</td>
<td>x y c @</td>
</tr>
<tr>
<td>@</td>
<td>•</td>
<td>b x y c @</td>
</tr>
<tr>
<td>?</td>
<td>ERROR (REDUCE (bxY))</td>
<td></td>
</tr>
</tbody>
</table>

Illustrative Examples

The following section will provide a variety of examples to display some of the aspects of context resolvable grammars.

Example 4.3

Grammar: 1. $S \rightarrow @x@y@$
          2. $A \rightarrow y@y@$
          3. $A \rightarrow y$

Matrix:  

<table>
<thead>
<tr>
<th>A</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>(\Rightarrow)</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>(\Leftarrow)</td>
<td>(&lt;)</td>
</tr>
<tr>
<td>y</td>
<td>(\Leftarrow)</td>
<td>(\Rightarrow)</td>
</tr>
</tbody>
</table>

Note: The production $S$ shows the end markers appended, the way the analyzer does in the parsing algorithm. They then become usable context for resolving conflicts if needed.
(1) The conflict $y \prec y$, $y \equiv y$ and $y \succ y$ can be resolved by:

\[ R_{\succ,1}^2 (y,y) = \{y\}, R_{\equiv,2}^2 (y,y) = \{yA\}, R_{\prec,3}^2 (y,y) = \{yy\} \]

Since the individual sets are unique the conflict is resolvable on the right such that

Resolve $(y,y) = (\emptyset, yy, \prec, +)$

Resolve $(y,y) = (\emptyset, yA, \equiv, +)$

Resolve $(y,y) = (\emptyset, y\emptyset, \succ, \emptyset)$

For the conflict $A \succ y$ and $A \equiv y$:

\[ L_{\equiv,1}^2 (A,y) = \{xA\}, L_{\succ,1}^2 (A,y) = \{y\} \]

Since the left sets are unique the conflict is left resolvable where

Resolve $(A,y) = (xA, \emptyset, \equiv, +)$

Resolve $(A,y) = (yA, \emptyset, \succ, \emptyset)$

(3) Since the grammar contains a pair $(y,y)$ which is of order $(1,2)$ and the pair $(A,y)$ of order $(2,1)$; the grammar is of order $(2,2)$.

(4) A parse of the string $x y y y y$ follows:
<table>
<thead>
<tr>
<th>S-stack</th>
<th>P</th>
<th>I-stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>@</td>
<td>&lt;•</td>
<td>x y y y @</td>
</tr>
<tr>
<td>@ x</td>
<td>&lt;•</td>
<td>y y y y @</td>
</tr>
<tr>
<td>@ x y</td>
<td>&lt;•</td>
<td>y y y @</td>
</tr>
<tr>
<td>@ x y y</td>
<td>&lt;•</td>
<td>y y @</td>
</tr>
<tr>
<td>@ x y y y</td>
<td>&gt;</td>
<td>y @</td>
</tr>
<tr>
<td>@ x y y</td>
<td>&lt;•</td>
<td>y y @</td>
</tr>
<tr>
<td>@ x y y</td>
<td>=</td>
<td>A y @</td>
</tr>
<tr>
<td>@ x y y A</td>
<td>&gt;</td>
<td>y @</td>
</tr>
<tr>
<td>@ x y y</td>
<td>=</td>
<td>A y @</td>
</tr>
<tr>
<td>@ x y</td>
<td>=</td>
<td>y A y @</td>
</tr>
<tr>
<td>@ x</td>
<td>&lt;•</td>
<td>y y A y @</td>
</tr>
<tr>
<td>@ x</td>
<td>=</td>
<td>A y @</td>
</tr>
<tr>
<td>@ x A</td>
<td>=</td>
<td>y @</td>
</tr>
<tr>
<td>@ x A y</td>
<td>&gt;</td>
<td>@</td>
</tr>
<tr>
<td>@ x A</td>
<td>=</td>
<td>y @</td>
</tr>
<tr>
<td>@ x</td>
<td>=</td>
<td>A y @</td>
</tr>
<tr>
<td>@</td>
<td>=</td>
<td>x A y @</td>
</tr>
<tr>
<td>@</td>
<td>&lt;•</td>
<td>S @</td>
</tr>
</tbody>
</table>

**Example 4.4**

**Grammar:**
1. \( E \rightarrow @T@ \)
2. \( E \rightarrow @E + T@ \)
3. \( T \rightarrow P \)
4. \( T \rightarrow T \cdot P \)
5. \( P \rightarrow (E) \)
6. \( P \rightarrow v \)
(1) **(1,1) Precedence relations**

<table>
<thead>
<tr>
<th>( U )</th>
<th>( LCD^{(1)}U )</th>
<th>( RCD^{(1)}U )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E )</td>
<td>( T, E, P, v, ( )</td>
<td>( T, P, ), v )</td>
</tr>
<tr>
<td>( T )</td>
<td>( P, T, v, ( )</td>
<td>( P, ) , v )</td>
</tr>
<tr>
<td>( P )</td>
<td>( v, ( )</td>
<td>( ), v )</td>
</tr>
</tbody>
</table>

| \( E \) | \( \neq \) | \( \neq \) |
| \( T \) | \( > \neq \) | \( > \) |
| \( P \) | \( > \neq \) | \( > \) |
| \( + \) | \( < \neq \) | \( < \neq \) | \( < \neq \) |
| \( * \) | \( = \neq \) | \( = \neq \) | \( = \neq \) |
| \( ( \) | \( < \neq \) | \( < \neq \) | \( < \neq \) |
| \( ) \) | \( \neq \) | \( \neq \) |

(2) For the conflict + \( * \rightarrow T \) and + \( \neq T \),

\[
L_{=, 2}^{2}(+, T) = \{ E+, T+, P+, v+, )+ \}
\]

\[
L_{<+, 2}^{2}(+, T) = \{ E+, T+, P+, v+, )+ \}
\]

since \( L_{=, 1}^{2}(+, T) \cap L_{<+, 1}^{2}(=, T) \neq \emptyset \) the pair is not left resolvable for \( k = 2 \).

\[
R_{<+, 2}^{2}(+, T) = \{ T*, P*, v*, (E, T, (P, v, ( (})
\]

\[
R_{=, 2}^{2}(+, T) = \{ T\emptyset \}
\]

Hence the pair \((+, T)\) is of order \((1,2)\) for this conflict.

(3) For the conflict \(\neq E \) and \( \neq E \),

\[
R_{=, 5}^{2}((, E) = \{ E \}, R_{<+, 5}^{2}((, E) = \{ E+ \}
\]

So the grammar is \((1,2)\) for this conflict.

And consequently the grammar as a whole is \((1,2)\).

(4) A parse of a legal string is given (note: the parse will be shown only at the pertinent phases similar to the method described in Chapter 2).
The previous example is the usual arithmetic expression grammar which contains productions which are left recursive. The example was worked to show the ability of context resolvable construction to handle the recursive property. It should be noted however, that a minimum number of conflict pairs \((A_i, A_j)\) is desirable, since the resolving table will be smaller and the extra parsing time needed for the table search can be eliminated. This can be accomplished in the case of recursive productions by using a transformation on the recursive productions. Example 4.4 will be displayed again using the transformation.

**Example 4.5**

Grammar: 

\[
S \rightarrow E, 
E \rightarrow E + T, 
E \rightarrow T, 
T \rightarrow Q, 
Q \rightarrow Q \times P, 
Q \rightarrow P, 
P \rightarrow (E), 
P \rightarrow v.
\]

(1) The precedence matrix is
which shows that no conflicts of precedence relations occur in the transformed grammar.

The following example is a grammar given by Wirth and Weber (13) which is not precedence \((m,n)\); it is also not context resolvable.

**Example 4.6**

Grammar: 1. \(A \rightarrow B\)
2. \(A \rightarrow (B)\)
3. \(B \rightarrow (b)\)
4. \(B \rightarrow b\)

(1) The precedence matrix is:

<table>
<thead>
<tr>
<th>B ( b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
</tr>
<tr>
<td>(</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>)</td>
</tr>
</tbody>
</table>

which is not precedence \((m,n)\); it is also not context resolvable.
(2) For the conflict (≤ b and (≠ b):

\[ L_{≤,1}^2((\cdot, b)) = \{\cdot\}, \quad L_{<,2}^2((\cdot, b)) = \{\cdot\} \] therefore, it is not left resolvable for \( k = 2 \).

\[ R_{≥,1}^3((\cdot, b)) = \{b\}_{\cdot}, \quad R_{<,2}^3((\cdot, b)) = \{b\}_{\cdot}, \{b\}_{\cdot} \] therefore, it is not right resolvable for \( k = 3 \).

(3) For the conflict b = ) and b •>):

\[ R_{≥,2}^2(b, )) = \{\), \) \} \] therefore, it is not right resolvable for \( k = 2 \).

and \( L_{<,3}^3(b, )) = \{\cdot\}_{\cdot}, \{\cdot\}_{\cdot} \) therefore, it is not left resolvable for \( k = 3 \).

(4) Since neither of the conflicts can be resolved and the value of \( k \) is large enough to include the end markers at both ends of the left and right sets, the grammar is not \((m, n)\) for any \( m \) or \( n \), since increasing \( k \) would produce null sets.

Another interesting grammar is one which Knuth (9) has shown to be LR(1). The example will show that the grammar is not a total precedence grammar; however, it is context resolvable.

**Example 4.7**

Grammar:

1. \( S \rightarrow aAd \)
2. \( S \rightarrow bAB \)
3. \( B \rightarrow d \)
4. \( A \rightarrow cA \)
5. \( A \rightarrow c \)

(1) The precedence matrix is:
Since the matrix contains conflicts, the grammar is not total precedence.

(2) For the conflict A •> B and A = B:
\[ L^2_{\neq, 1}(A, B) = \{bA\} \]
\[ L^2_{\neq, 2}(A, B) = \{cA\} \]
therefore, it is left resolvable with \( k = 2 \).

(3) For the conflict A <* d, A = d, and A •> d
\[ L^2_{\neq, 2}(A, d) = \{cA\} \]
\[ L^2_{<, 2}(A, d) = \{bA\} \]
\[ L^2_{\neq, 1}(A, d) = \{aA\} \]
Hence the context is left resolvable with \( k = 2 \).

(4) The resolve table is:
Resolving (A, B) = (bA, *, =, +)
Resolving (A, B) = (cA, *, •>, *)
Resolving (A, d) = (cA, *, •>, +)
Resolving (A, d) = (bA, *, <•, +)
Resolving (A, d) = (aA, *, =, *)

(5) A parse of the string @a c c c d @ is:
@ @ a c c c d @
SCAN @ [ a [ c [ c [ c ] ] ] ]
REDUCE @ [ a [ c [ c A ] ] ? d
RESOLVE @ [ a [ c [ c A ] ] d
REDUCE @ [ a [ c A ] ? d
RESOLVE @ [ a [ c A ] ] d
REDUCE @ [ a A ] ? d
RESOLVE @ [ a A ] d ] @
REDUCE @ [ S ] @
SUCCESS

(6) A parse of the string @ b c c c d @ will be:
@ b c c c c d
SCAN @ [ b [ c [ c [ c [ c ] ] ] ] d
REDUCE @ [ b [ c [ c [ c A ] ] ] d
RESOLVE @ [ b [ c [ c A ] ] d
REDUCE @ [ b [ c A ] ? d
RESOLVE @ [ b [ c A ] ] d
REDUCE @ [ b A ] ? d
RESOLVE @ [ b A ] d ] @
REDUCE @ [ b A ] ? B ] @
RESOLVE @ [ b A ] B ] @
REDUCE @ [ S ] @
SUCCESS

The following example contains an ambiguous grammar which is not context resolvable.
Example 4.8

Grammar: 1. \( Z \rightarrow X \)
2. \( X \rightarrow a \)
3. \( X \rightarrow Xb \)
4. \( X \rightarrow Va \)
5. \( V \rightarrow Y \)
6. \( Y \rightarrow YdY \)
7. \( Y \rightarrow e \)

(1) The precedence relations for the above grammar show the following conflicts:

(a) \( Y \rightarrow d \) and \( Y \equiv d \)
(b) \( d \leftarrow Y \) and \( d \equiv Y \)

(2) The contextual sets for (a) would be

(a) \( R_{2,6}^{-2}(Y,d) = \{dY, de\} \) and \( R_{2,6}^{+2}(Y,d) = \{dY, de\} \)

Since addition right context would be obtained from the \( Y \) in both R-sets, the additional context will not generate unique R-sets for any \( k \).

(b) \( L_{2,6}^{-2}(d,Y) = \{dY\} \) and \( L_{2,6}^{+2}(d,Y) = \{dY, de\} \)

The left context for the \( L_{2,6}^{-2}(Y,Y) \) set is null at \( k = 2 \) so the conflict is not left resolvable for any \( k \).

(c) Since the left and right sets are not unique for any \( k \), implied by (a) and (b), the grammar is not context resolvable.
DISCUSSION

The definition of Context Resolvable grammars is structured as a subset of the context free grammars which contains other precedence grammars. The inclusion diagram in Figure 3.1 shows the related sets.

The context resolvable algorithm parses strings generated by a grammar in a time bound which is proportional to the length of the string, as shown by Colmerauer (2). It applies a minimal reduction for all Context Resolvable grammars which are of order (1,1) and uses the resolve table only for those symbol pairs which are in conflict.

The use of this partitioning results in the properties which keep the parsing information to a minimum. In addition, it should be possible to define current programming languages in this class of grammars. Colmerauer (2) has implemented a compiler for a slightly modified algol using total precedence relations. McKeeman's work described in Feldman and Gries (6) using "triples" has implemented XPL, a very PL/I like language. These make the context resolvable grammars a good candidate for automatic compiler techniques. Using a defining set of productions as input, the context resolvable algorithm can identify those pairs \((A_i, A_j)\) which are total precedence of order (1,1) and then compute the resolution list for those pairs in conflict. This provides identification of those productions or features in the language which are "inefficient", and could be changed or deleted.

When defining standard languages by context resolvable grammars conflict productions isolate possible areas for lexical solution to provide for faster compilation.
The elements of the parsing algorithm are given in somewhat general terms because specific details are usually machine related. The algorithm parses via information lists which are precedence related and do not carry extraneous information as do some of the tree structure types which carry all possible parse combinations.

The algorithm can also be useful for extensible languages. The matrix information however, must be retained. The efficiency of the matrix type parser for extensible languages is that only relations for each new production as it is added need be computed. Where a conflict arises from new insertions, the context resolvable definition handles this normally and total recompilation is not required.

As with previous precedence techniques, semantic actions are easily tied in at any required point by relation to that production which is being reduced. An important factor in context resolvable grammar construction is that some production sequences are more translatable to a machine code that a sequence necessarily required to eliminate precedence conflicts. The production forms more translatable to the machine language can be handled simply as context resolvable.

This leads into the area of what constitutes a good production set for a particular class of grammars. Certain production forms are ill defined and transformations ought to be classified which could eliminate specific "problem" types. A question to be answered for language definitions is what constitutes a good set of productions. Is a linear tree form like EULER (13) a good set? Are productions in tact the way to define language syntax? One of the important points of many of the
algorithms (possible all) is that they are related only to the grammar and not to the entire language. One of the possible areas of investigation should be ways of constructing parsers on other methods of language definitions. Grammars might be investigated to the extent that language related productions could be classified, and technique developed for minimizing the number of productions needed to specify a specific language.

An area specific to context resolvable grammars, is the class of grammars it contains. A definite area for further research is to try to extend the context resolvable grammars to include all of the deterministic languages. It also seems possible that this extension might include all the context free grammars.
BIBLIOGRAPHY


ACKNOWLEDGMENTS

I would first like to express my appreciation to my wife Joy and family for their endless patience and many sacrifices. To Professor Robert Stewart for his guidance and supervision my sincerest gratitude and thanks. I also wish to express my thanks to Professor Roy Keller for his extensive critique and assistance in structuring this thesis. My thanks also to Professor Clair Maple for his confidence and support of my program. My sincere thanks to Professors Robert Lambert and Arthur Pohm for the courses they taught, and friendship they extended. I would also like to extend a special acknowledgment and thanks to Mr. Robert Sharpe (now with the Xerox Corporation) for his wonderful teaching and encouragement which helped through all of my studies. Also, a special thanks to Dr. Charles Wright for his suggestions.

I would also like to thank Mrs. Betty Dahlgren for her many efforts in the typing of the rough drafts. To my friend, Virgil Wallentine, sincerest appreciation and thanks for the numerous discussions we've had and the editing of the final copy. I would also like to thank Mrs. Dee Wallentine for the final typing of the manuscript.