Precedence grammars for compiler construction

Jennifer Fan-Yuan Tseng

Atlanta University
PRECEDENCE GRAMMARS
FOR COMPILER CONSTRUCTION

A THESIS
SUBMITTED TO THE FACULTY OF ATLANTA UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF SCIENCE

BY
JENNIFER FAN-YUAN TSENG.

DEPARTMENT OF MATHEMATICAL SCIENCES
ATLANTA, GEORGIA
DECEMBER 1981
ABSTRACT

COMPUTER SCIENCE

TSENG, JENNIFER FAN-YUAN

B.S. PROVIDENCE COLLEGE, 1976

PRECEDENCE GRAMMARS OF COMPILER CONSTRUCTION.

Advisor: Professor Nazir A. Warsi

Thesis dated December, 1981

The basic objective of this thesis is to study the theory of precedence grammars. In particular, this paper deals with results concerning LR(k), SPG, UIEPG, and UIWPG and their hierarchical structures. Various theorems are proved to show different relationships. A brief theory of precedence parsing is also presented.
ACKNOWLEDGEMENTS

I wish to sincerely thank Dr. Nazir A. Warsi for his guidance, discriminative reading, and valuable time contributed to the completion of this paper. Also, I wish to thank Dr. Benjamin Martin, Dr. B. Setzer, and other faculty members of the Mathematical Sciences Department for their valuable help.
# TABLE OF CONTENTS

I. INTRODUCTION ........................................ 1

II. LR(k) PARSING ........................................ 7
   \hspace{1em} Shift-Reduce Parsing
   \hspace{1em} LR(k) Grammar

III. SIMPLE PRECEDENCE GRAMMARS .................... 11
   \hspace{1em} Definition of Precedence Relations
   \hspace{1em} Simple Precedence Parsing
   \hspace{1em} Comparison of SPG and LR(k)

IV. WEAK PRECEDENCE GRAMMARS ...................... 20
   \hspace{1em} Definition of Weak Precedence Grammar
   \hspace{1em} Relation between SPG and UIWPG

V. EXTENDED PRECEDENCE GRAMMARS ................. 23
   \hspace{1em} Definition of Extended Precedence Grammar
   \hspace{1em} Relation between UIEPG and LR

VI. CONCLUSION ........................................ 26
   \hspace{1em} Hierarchical Relations between Precedence Grammars
   \hspace{1em} Language Hierarchies
   \hspace{1em} Comparison of Precedence and LR Parsing Techniques

BIBLIOGRAPHY .......................................... 28

GLOSSARY .................................................. 30
CHAPTER I

INTRODUCTION

Introductory Remarks

The paper attempts to discuss the theory of precedence grammars with particular emphasis on the hierarchical relations of subclasses of LR grammars. In particular the following objectives are met:

1. We formulate and discuss the basic theory of Simple Precedence Grammar (SPG), Uniquely Invertible Extended Precedence Grammar (UIEPG), and Uniquely Invertible Weak Precedence Grammar (UIWPG). To bring out the theory clearly and to be able to use it for hierarchical discussion we have introduced some new notations, rearranged the material and have presented the theory in the form of theorems. We have provided original proofs for such theorems.

2. We have established several results proving the various hierarchical relations of precedence grammars as subclasses of CFG and LR grammars. Most of the material presented in Chapters III-V was suggested by exercises from {AU1}, p. 314. The material has be reorganized using mathematic symbolism and formalism. Original proofs are provided as far as possible.

3. A brief summary of hierarchies of precedence languages and their interrelations is provided without any proofs.

4. A brief critical comparison of LR and precedence parsing techniques is given.
In a given compiler, the output for a lexical analyzer is a string of tokens. This string constitutes the input to the syntactic analyzer which examines only the first components of the token. The second component is used later in the compiling process to generate the machine code.

Compilers for various languages have a certain process in common. Parsing is a process in which the string obeys certain structural conventions explicit in the syntactic structure of the language. For example, the syntactic structure of the expression $A + B \times C$ must reflect that $B$ and $C$ are first multiplied and then the result is added to $A$. No other ordering of the operations will produce the desired result.

From a set of syntactic rules it is possible to automatically construct parsers which make sure that a source program obeys the syntactic structure defined by these syntactic rules. It has two cases top-down parsing and bottom-up parsing.

**Shift-Reduce Parsing**

One bottom-up method of parsing is shift-reduce parsing. A parsing method is bottom-up if it attempts to construct a parse tree for an input string beginning at the leaves (bottom) and working up towards the root (top). At each step a string matching the right side of a production is replaced by the symbol on the left.

Let us consider a grammar with productions $S \rightarrow AB$, $A \rightarrow ab$, and $B \rightarrow aba$. Let ababa be an input string. First $a$ is shifted onto a pushdown list. Since no reduction is possible, $b$ is shifted onto the pushdown list and $ab$ is replaced on the top of the pushdown list by $A$.

As $A$ cannot be further reduced, $a$ is shifted onto the pushdown list. Again no reduction is possible, so $b$ is shifted onto the pushdown list and it is found that no
reductions are possible. Then a backtrack is made to the last position where the
pushdown list contained Aab and ab is replaced by A. Since again no reduction is
possible, we shift a onto the pushdown list. The pushdown list now contains Aaba.
We reduce aba to B. Next we replace AB by S and the process is finished.

\[
\begin{align*}
A & \rightarrow \text{aba} \\
& \quad A \\
& \quad a \quad b \\
& \quad A \\
& \quad a \quad b \\
& \quad A \\
& \quad a \quad b \\
& \quad a \quad b \\
& \quad S \\
& \quad A \\
& \quad a \quad b \\
& \quad B \\
& \quad a \quad b \\
\end{align*}
\]

Operator-Precedence Grammars

For a certain small class of grammars we can easily construct an efficient
shift-reduce parser. It is an easy-to-implement parsing technique called operator-
precedence parsing.

In operator-precedence parsing, we use three disjoint precedence relations
which guide the selection of a handle. If \( a \prec b \), we say a "yields precedence to"
b; if \( a \sim b \), a "has the same precedence as" b; if \( a \succ b \), a "takes precedence over"
b. The use of the precedence relations is to delimit the handle of a right-sentential form, with \( < \) marking the left end, \( \equiv \) appearing in the interior of the handle, and \( \Rightarrow \) marking the right end.

For a certain small class of grammars we can easily construct efficient shift-reduce parsers by hand. These grammars have the property that the right side of no production has two adjacent nonterminals. A grammar with this property is called an operator grammar. If we use an unambiguous operator grammar for arithmetic expressions, we can construct a reliable operator-precedence table for the grammar.

Let \( G \) be an \( \epsilon \)-free operator grammar. For each two terminal symbols \( a \) and \( b \), the operator relations are defined as follows:

1. \( a < b \) if for some nonterminal \( A \) there is a right side of a production of the form \( a a A \beta \) , and \( A \rightarrow^+ rb \), where \( r \) is either \( \epsilon \) or a simple nonterminal. That is, \( a < b \) if a nonterminal \( A \) appears immediately to the right of \( a \) and derives a string in which \( b \) is the first terminal symbol. Also, define \( \$ < b \) if there is a derivation \( S \rightarrow^+ rb \) where \( r \) is \( \epsilon \) or a simple nonterminal.

2. \( a \equiv b \) if there is a right side of a production of the form \( a a \beta r b \), where \( \beta \) is either \( \epsilon \) or a simple non-terminal. That is, \( a \equiv b \) if \( a \) appears immediately to the left of \( b \) in a right side, or if they appear separated by one nonterminal.

3. \( a \Rightarrow b \) if for some nonterminal \( A \) there is a right side of a production of the form \( a A \beta \) with \( A \rightarrow^+ ra \delta \), where \( \delta \) is either \( \epsilon \) or a single nonterminal. That is, \( a \Rightarrow b \) if a nonterminal appearing immediately to the left of \( b \) derives a string whose last terminal symbol is \( a \). Also, we define \( a \Rightarrow \$ \) if \( S \Rightarrow ra \delta \) where \( \delta \) is either \( \epsilon \) or a single nonterminal.

These definitions can be pictured as follows:
If $a \triangleright b$, then by rule 3, it appears that $a$ is the rightmost terminal of the handle. The relation $a \ll b$ indicates that $b$ is the leftmost terminal of a handle.

Definition 1:

An operator-precedence grammar is an $\varepsilon$-free operator grammar in which the precedence relations $\ll$, $\ll$, and $\triangleright$ are disjoint. In other words, for any pair of terminals $a$ and $b$, at most one of the relations $a \triangleright b$, $a \ll b$, and $a \ll b$ may hold.

Example 1:

Consider the following grammar:

\[
E \rightarrow E + T | T \\
T \rightarrow T \ast F | F \\
F \rightarrow (E) | \text{id}
\]
First, to compute the $=$ relation, we look for right sides with two terminal. Only one right side, $(E)$ qualifies. So we determine that $(\varnothing)$. Next, consider $\triangleleft$. We look for right sides with a terminal immediately to the left of a nonterminal to play the roles of $a$ and $A$ in rule 2. For each such pair, $a$ is related by $\triangleleft$ to any terminal which is a prefix of a string derivable from $A$. These candidates for the grammar are $+$ and $T$ in the right side $E + T$, $*$ and $F$ in $T * F$, and $(E)$ (and $E$ in $(E)$). The first of these gives $+ \triangleleft *, + \triangleleft ( +$, and $+ \triangleleft \u2009\text{id}$. The $*$ and $F$ give $* \triangleleft ( +$, and $* \triangleleft \u2009\text{id}$. The $(E)$ give $(E) \triangleleft *, (E) \triangleleft +$, $(E) \triangleleft (,$ and $(E) \triangleleft \u2009\text{id}$. We then add the relationships $\$$ \triangleleft *, \$$ \triangleleft +, \$$ \triangleleft (, \text{and } \$$ \triangleleft \u2009\text{id}$, since $\$$ must be related by $\triangleleft$ to all possible first terminals derivable form the start symbol $E$.

Similarly, we can construct the $\triangleright$ relation. We look for the right sides with a nonterminal immediately to the left of a terminal to play the roles of $a$ and $b$ in rule 3. Every terminal that could be the last in a string derivable from $A$ is related by $\triangleright$ to $b$. So the pairs corresponding to $A$ and $b$ are $E$ and $+$, and $E$ and $\)$. Thus, we have the relations: $* \triangleright +, + \triangleright +, ( \triangleright +, \text{id} \triangleright +, * \triangleright *, ) \triangleright *, \text{id} \triangleright *, * \triangleright (, + \triangleright ), ) \triangleright ), \text{and id} \triangleright \)$. We add the relations $* \triangleright \$$, $+ \triangleright \$$, $) \triangleright \$$, and $\text{id} \triangleright \$$ according to rule 3. Thus we have the following table:

<table>
<thead>
<tr>
<th>$+$</th>
<th>$*$</th>
<th>$($</th>
<th>$)$</th>
<th>$\text{id}$</th>
<th>$$$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\triangleright$</td>
<td>$\triangleleft$</td>
<td>$\triangleleft$</td>
<td>$\triangleleft$</td>
<td>$\triangleleft$</td>
<td>$\triangleleft$</td>
</tr>
</tbody>
</table>

This table represents the relationships between the symbols based on the rules provided.
CHAPTER II

LR(K) PARISING

This chapter deals with the construction of an efficient bottom-up parser for a large class of context-free grammars (CFG). These parsers are called LR parsers because they scan the input from left-to-right and construct a rightmost derivation in reverse.

An LR parser has an input, a stack, and a parsing table. The input is read from left-to-right, one symbol at a time. The stack contains a string of the form $s_0 X_1 s_1 X_2 s_2 ... X_m s_m$, where $s_m$ is on top. Each $X_i$ is a grammar symbol and $s_i$ summarizing the information contained in the stack below it and is used to guide the shift-reduce decisions.

Shift-Reduce Parsing

Shift-reduce parsing consists of shifting input symbols onto a pushdown list until a handle appears on top. The handle is then reduced. If no errors occur, this process is repeated until all of the input string is scanned and only the sentence symbol appears on the pushdown list.

A shift-reduce parsing algorithm can be considered as a program for an extended deterministic pushdown transducer which parses bottom-up.

There are three decisions for a shift-reduce parsing algorithm. The first is to determine before each move whether to shift an input symbol onto the pushdown list or to call for a reduction. The second and the third decisions occur after the right
end of a handle is located. Once the handle is known to lie on top of the pushdown list, the left end of the handle must be located within the pushdown list. Then the appropriate nonterminal is replaced.

A grammar in which no two distinct productions have the same right side is said to be uniquely invertible. If a grammar is uniquely invertible, then there is exactly one nonterminal by which a right side of production can be replaced.

Example 2:

Consider a grammar with the following productions:

1. \( S \rightarrow SaSb \)
2. \( S \rightarrow e \)

Their rightmost derivation is \( S \rightarrow SaSb \rightarrow SaSaSbb \rightarrow SaSabb \rightarrow Saabb \rightarrow aabb \). Let us parse the sentence aabb using a pushdown list and a shift-reduce parsing algorithm. We use $ as an endmarker for both the input string and the bottom of the pushdown list.

The algorithm will make the following sequence of moves.

\[
\begin{align*}
(\$,aabb\$,e) &\rightarrow (\$,aabb\$,2) \\
&\rightarrow (Sa,\$,2) \\
&\rightarrow (SaS,\$,22) \\
&\rightarrow (SaSa,\$,22) \\
&\rightarrow (SaSaS,\$,222) \\
&\rightarrow (SaSaSb,\$,222) \\
&\rightarrow (SaSb,\$,2221) \\
&\rightarrow (SaSb$,\$,22211) \\
&\rightarrow (S$,\$,22211) \\
&\rightarrow accept.
\end{align*}
\]
LR(k) Grammar

Another source of information, besides LR parser, which helps in making the shift-reduce decisions is about the next \( k \) input symbols. A grammar that can be parsed by an LR parser examining up to \( k \) input symbols on each move is called an LR(k) grammar.

We say that a grammar is LR(k) if given a rightmost derivation \( S = a_0 \rightarrow^* a_1 \rightarrow^* \ldots \rightarrow^* a_m = Z \), we can isolate the handle of each right-sentential form and determine which nonterminal is to replace the handle by scanning \( a_1 \) from left to right, but only going at most \( k \) symbols past the right end of the handle of \( a_1 \). At each step the lookahead string will consists only of \( k \) or fewer terminal symbols.

We will introduce an augmented grammar first before defining an LR(k) grammar formally.

Definition 2:

Let \( G = (N, \Sigma, P, S) \) be a context-free proper grammar. We define the augmented grammar derived from \( G \) as \( G' = (N \cup \{ S' \}, \Sigma, P \cup \{ S' \rightarrow S \}, S') \). The augmented grammar \( G' \) is merely \( G \) with a new starting production \( S' \rightarrow S \), where \( S' \) is a new start symbol not in \( N \). We assume that \( S' \rightarrow S \) is the zeroth production in \( G' \) and that the other productions of \( G \) are numbered 1, 2, ..., \( p \). We add the starting production so that when a reduction using the zeroth production is called for, we can interpret this reduction as a signal to accept.

Definition 3:

Let \( G = (N, \Sigma, P, S) \) be a CFG and let \( G' = (N', \Sigma, P', S') \) be its augmented grammar. We say that \( G \) is LR(k), \( k > 0 \), if whenever the following three conditions are satisfied:
Then $A = r$, $B = A$, and $x = y$. Here, $A$, $B$ are nonterminals. A grammar is LR if it is LR$(k)$ for some $k > 0$.

In constructing a deterministic right parser for each LR$(k)$ grammar $G = (N, \Sigma, P, S)$, we go through the following steps. First, the parser will be constructed from the augmented grammar $G'$. The parser will behave like the shift-reduce parser except that the LR$(k)$ parser will put special information symbols, called LR$(k)$ tables, on the pushdown list above each grammar symbol. These LR$(k)$ tables will determine whether a shift or a reduce move is to be made. In the case of a reduce move, which production is to be used is decided.

There are many different parsing tables that can be used in an LR$(k)$ parser for a given grammar. Some parsing tables may detect errors sooner than others, but they all accept the sentences generated by the grammar. Three different techniques for producing LR parsing tables are described. The first method, called a simple precedence grammar (SPG for short), is easiest to implement. The second method is called an extended precedence grammar (EPG for short). The third method is called a weak precedence grammar (WPG for short).
CHAPTER III

SIMPLE PRECEDENCE GRAMMARS

The simplest shift-reduce algorithms are based on precedence relations. In a precedence grammar the location of the boundaries of the handle for a right-sentential form. The key to the precedence parsing is the definition of a precedence relation $\Rightarrow$ between grammar symbols such that on scanning from left to right of a right-sentential form $\alpha \beta w$ with handle $\beta$, the precedence relation $\Rightarrow$ is first found to hold between the last symbol of $\beta$ and the first symbol of $w$. For the shift-reduce parsing algorithm, the decision to reduce will occur whenever the precedence relation $\Rightarrow$ holds between the top of the pushdown list and the first of remaining input symbols. If the relation $\Rightarrow$ does not hold then a shift will be called.

The simple precedence parsing technique uses three precedence relations $\prec$, $\preceq$, and $\Rightarrow$ to isolate the handle in a right-sentential form $\alpha \beta w$. The grammar is a simple precedence grammar if it has no $\epsilon$-productions, no two productions have the same right sides, and the relations $\prec$, $\preceq$, and $\Rightarrow$ are disjoint. If $\beta$ is the handle, then relation $\prec$ or $\preceq$ holds between any pair of symbols in $\delta$, $\preceq$ hold between suffix of $\delta$ and prefix of $\beta$, $\preceq$ holds between any pair of symbols in the handle itself, and the relation $\Rightarrow$ holds between the last symbol of $\beta$ and the first of $w$. So the handle of a right-sentential form of a simple precedence grammar can be located by scanning the sentential form from left to right until the precedence relation $\Rightarrow$ is found. Then symbols are scanned to the left until the first $\preceq$ is found. The handle is the string between $\prec$ and $\Rightarrow$. This process can be repeated until the input string is either reduced to the starting symbol or no further reductions are possible.
Definition of Precedence Relations

Precedence relations lie at the heart of the theory of precedence parsing. These relations commonly known as Wirth-Weber relations are defined as follows.

Definition 4:

For a CFG $G$, the precedence relations $\prec$, $\preceq$, and $\succ$ are defined by

1. $S \prec X$ iff $S \rightarrow X_\alpha$ and $Y \succ S$ iff $S \rightarrow \alpha Y$
2. $X \preceq Y$ iff $\exists A \rightarrow \alpha X B_\beta$ in $P$ with $B \rightarrow \gamma Y_\rho$
3. $X \preceq Y$ iff $\exists A \rightarrow \alpha X Y B_\beta$ in $P$
4. $X \succ \alpha$ iff $\exists A \rightarrow \alpha B Y_\beta$ in $P$ with $Y \rightarrow \alpha \delta$ and $B \rightarrow \epsilon r X$

Agreement 1:

$G$ is a proper CFG without $\epsilon$-productions.

Definition 5:

A proper CFG without any $\epsilon$-production is called a precedence grammar if at most one precedence relation exists between any two symbols. In addition, if $G$ is also uniquely invertible then the grammar is called simple precedence grammar (SPG for short).

Definition 6:

$\text{Suf}_n(\alpha) =$ last $n$ symbols of $\alpha$

$\text{Pref}_n(\alpha) =$ first $n$ symbols of $\alpha$. 
Theorem 1: \((\{AU1\} \text{ p. 406})\)

Let \(G\) be a proper CFG without \(\varepsilon\)-productions. Suppose \(S \xrightarrow{\text{rm}} aAw \xrightarrow{\text{rm}} a\beta w\). Then any two adjacent symbols of \(a\beta\) are related either by \(\preceq\) or by \(\preceq\) with \(\text{Suf}_1(a) \preceq \text{Pref}_1(\beta)\).

Proof:

We establish the result by induction on number of steps in the derivation of a right-sentential form. For the first step consider the derivation \(S \xrightarrow{\text{rm}} a\beta w\). By definition \(a\beta w = a_1A_1w_1\) where \(S \xrightarrow{\text{rm}} a\beta w = a_1A_1w_1 \xrightarrow{\text{rm}} a_1\beta_1w_1\). We show that the result also holds for \(a_1\beta_1w_1\). Since \(a_1\) is included in \(a\beta\), either \(\preceq\) or \(\preceq\) holds between \(\text{Suf}_1(a_1)\) and \(A_1\).

Case 1: \(\text{Suf}_1(a_1) = A_1\)

There exists a production \(C \xrightarrow{} \text{Suf}_1(a_1)A_1\delta_1\) with \(A_1 \xrightarrow{} \beta_1\). Thus we have \(\text{Suf}_1(a_1) \preceq \text{Pref}_1(\beta_1)\).

Case 2: \(\text{Suf}_1(a_1) \preceq A_1\)

There is a production \(C \xrightarrow{} \text{Suf}_1(a_1)\beta_1\xi\) with \(B \xrightarrow{} A_1\gamma\). Since \(A_1 \xrightarrow{} \beta_1\), we have \(B \xrightarrow{} \beta_1\gamma\) and hence \(\text{Suf}_1(a_1) \preceq \text{Pref}_1(\beta_1)\). Since \(a_1\) is a prefix of \(a\beta\) and \(\beta_1\) is a handle, the assertion follows for \(a_1\beta_1w\).

Theorem 2: \((\{AU1\} \text{ p 406})\)

Let \(G\) be a proper CFG without \(\varepsilon\)-productions. Then \(\text{Suf}_1(\beta) \xrightarrow{} \text{Pref}_1(w)\) if \(SS \xrightarrow{\text{rm}} aAw \xrightarrow{\text{rm}} a\beta w\).
Proof:

We prove the result by induction again. For the first step, $S \Rightarrow_{rm} \beta \Rightarrow$. Then by definition $Suf_1(\beta) \Rightarrow \Rightarrow$. Now assume that the theorem holds for a right-sentential form $\alpha \beta w = \alpha_1 A_1 w_1 w$, with handle $\beta$ and that $S \Rightarrow_{rm} \alpha \beta w = \alpha_1 A_1 w_1 w \Rightarrow_{rm} \alpha_1 \beta_1 w_1 w$.

Case 1: $w_1 \neq \epsilon$

By induction hypotheses $Suf_1(\beta) \Rightarrow Pref_1(w)$. But $Suf_1(\beta) = A_1$ and $Pref_1(w) = Pref_1(w_1 w)$ since $w_1 = \epsilon$. Hence $A_1 \Rightarrow Pref_1(w_1 w)$. Therefore, there is a production $C \Rightarrow A1 \Rightarrow Pref_1(w_1 w)$ since $A_1 \Rightarrow Pref_1(w_1 w)$. Since $A_1 \Rightarrow \beta_1$, and $X \Rightarrow Pref_1(w_1 w)$, consequently $Suf_1(\beta_1) \Rightarrow Pref_1(w_1 w)$.

Case 2: $w_1 \neq \epsilon$

We have the following subcases:

1. If $A_1$ and $w_1$ both are in $\beta$ then $A_1 = Pref_1(w_1)$ by induction hypothesis. Then $\exists$ a production $C \Rightarrow A1 \Rightarrow Pref_1(w_1)$ with $A_1 \Rightarrow \beta_1$. Hence $Suf_1(\beta_1) \Rightarrow Pref_1(w_1) = Pref_1(w_1 w)$.

2. If $A_1$ is in $\alpha$ but $w_1$ is in $\beta$ then $A_1 \Rightarrow Pref_1(w_1)$ by induction hypothesis. Then $\exists$ a production $C \Rightarrow A1 \Rightarrow Pref_1(w_1)$ with $A_1 \Rightarrow \beta_1$. Hence, $Suf_1(\beta_1) \Rightarrow Pref_1(w_1 w)$.

3. If $A_1$ and $Pref_1(w_1)$ both lie in $\alpha$, then either $A_1 = Pref_1(w_1)$ or $A_1 \Rightarrow Pref_1(w_1)$ by induction hypothesis. The result follows as in 1 and 2.

These theorems shows that the precedence relation $\ll$ occurs at the beginning of a handle in a right-sentential form, $\Rightarrow$ holds between adjacent symbols of a handle, and $\Rightarrow$ holds at the right end of the handle. This is for all grammars with no $\varepsilon$-
productions. But in a precedence grammar there is at most one precedence relation between any pair of symbols in a viable prefix of a right-sentential form.

**Simple Precedence Parsing**

**Theorem 3:** ([AU1] p. 406, 407)

Let G be a CFG with $SS \xrightarrow{r_m} \alpha \Gamma w \xrightarrow{r_m} \alpha \beta w$. Then the handle is determined by $\prec \beta \succ$ with any two adjacent symbols of $\beta$ related by $\prec$ and those in $\alpha$ related by either $\preceq$ or $\ll$. 

**Proof:**

It is obvious from previous two theorems.

We shall describe how a deterministic right parser can be constructed for a simple precedence grammar.

**Algorithm 1:**

Let the productions be numbered from 1 to p.

1. The shift-reduce parsing algorithm will add $\$ as a bottom marker for the pushdown list and a right end marker for the input.
2. The shift-reduce function $f$ is independent of the pushdown list contents except for the topmost symbol and is independent of the remaining input except for the leftmost input symbol. So we define $f$ on $(N \cup \Sigma \cup \{ \$ \}) \times (\Sigma \cup \$$), except in the Case 3 below.
   - Case 1. $f(X, a) = \text{shift if } X \ll a \text{ or } X \preceq a$;
   - Case 2. $f(X, a) = \text{reduce if } X \gg a$;
   - Case 3. $f(S, \$) = \text{accept}$,
   - Case 4. $f(X, a) = \text{error, otherwise.}$
All these rules also can be applied by the precedence matrix itself.

3. The reduce function $g$ depends only on top of the pushdown list up to one symbol below the handle. The remaining input does not affect $g$. So we can define $g$ on $(\mathbb{N} \cup \mathbb{Z} \cup \{\$\})^*$ as follows:

a. $g(X_{k+1}X_kX_{k-1} \ldots X_1, e) = i$ if $X_{k+1} \prec X_k, X_{j+1} \leq X_j$ for $k > j > 1$, and production $i$ is $A \rightarrow X_kX_{k-1} \ldots X_1$. The reduction function $g$ is only applied when $X_1 \Rightarrow a$, where $a$ is the current input symbol.

b. $g(a, e) = \text{error}$, otherwise.

Example 3:

Let $G$ be a CFG with following productions:

1. $S \rightarrow aSSb$
2. $S \rightarrow c$

The precedence relations for $G$ are given by

$$
\begin{array}{cccc}
S & a & b & c & $ \\
S & \prec & \prec & \prec & \prec \\
a & \prec & \prec & \prec & \\
b & \succ & \succ & \succ & \succ \\
c & \succ & \succ & \succ & \succ \\
$ & \prec & \prec & \prec & \\
\end{array}
$$

We can use the precedence matrix for the shift-reduce function $f$ and the reduce function $g$ as follows:
1. \( g(XaSSb) = 1 \) if \( X \in \{ S, a, $ \} \)

2. \( g(Xc) = 2 \) if \( X \in \{ S, a, $ \} \)

3. \( g(a) = \) error, otherwise.

With input accb, we have the following sequence of moves.

\[
\begin{array}{l}
($) , accb$, e) \rightarrow (_S \rightarrow ($a, ccb$, e) \\
\rightarrow (_S \rightarrow ($ac, cb$, e) \\
\rightarrow (_) \rightarrow ($aS, cb$, 2) \\
\rightarrow (_S \rightarrow ($aSc, b$, 2) \\
\rightarrow (_) \rightarrow ($aSS, b$, 22) \\
\rightarrow (_S \rightarrow ($aSSb, $, 22) \\
\rightarrow (_) \rightarrow ($S, $, 221)
\end{array}
\]

This algorithm is an example of general shift-reduce algorithm with decisions of shift-reduction functions depending upon top symbol of the stack and the next input symbol. $ is the bottom marker of the stack as well as the right end marker of input string. A table of precedence relations is consulted for the relation between top of the stack and the next input. A shift is made as long as \( \# \) or \( \langle \) encountered, and until the first \( \rangle \) is reached. Then the part of the string enclosed between this \( \rangle \) on the right and the nearest \( \langle \) on the left gives the handle. Thus reduction is made, and the symbol pushed onto the top. This process is continued until either an error or an accept state is reached.

Comparison of SPG and LR(k)

Theorem 4: \( \text{SPG} \neq \text{LR}(k) \) (p. 410, 424)

Proof:

Let \( G' \) be the augmented grammar corresponding to SPG \( G \), and let \( S' \longrightarrow_{rm}^* \alpha Aw \longrightarrow_{rm} a\beta w, S' \longrightarrow_{rm}^* rBx \longrightarrow_{rm} a\beta y \) with \( \text{FIRST}_1(w) = \text{FIRST}_1(y) \). If \( x \neq y \) then we
have two cases.

Case 1. \( x_1 \neq \varepsilon, y = x_1 x \) and \( B \rightarrow \beta_1 x_1 \) was used to derive \( y \). If \( \beta_1 = \varepsilon \), \( S' \xrightarrow{r_m} rBx \xrightarrow{r_m} r_1 x_1 y = \alpha \beta y \) so \( r = \alpha \beta \). Then by Theorem 1, \( Suf_1(r) \triangleleft Pref_1(x_1) = Pref_1(y) \). But since \( S' \xrightarrow{r_m} \alpha Aw \xrightarrow{r_m} \alpha \beta w \), we have \( Suf_1(r) = Suf_1(\alpha \beta) = Suf_1(\beta) \triangleright Pref_1(w) = Pref_1(y) \). This is a precedence conflict. If \( \beta_1 \neq \varepsilon \), then \( Suf_1(\beta_1) = Pref_1(x_1) = Pref_1(y) \). But \( Suf_1(\beta_1) = Suf_1(\beta) \triangleright Pref_1(x_1) = Pref_1(y) \).

Case 2. \( y_1 \neq \varepsilon, x = y_1 y \) and \( B \rightarrow \beta_1 \) was used to derive \( \alpha \beta y \). Then \( S' \xrightarrow{r_m} rBy \xrightarrow{r_m} r_1 y_1 y = \alpha \beta y \) so that \( r_1 y_1 = \alpha \beta \). Hence by Theorem 1, since any two adjacent symbols in \( \alpha \beta \) are either related by \( = \) or \( \preceq \), we must have \( Suf_1(\beta_1) = Pref_1(y_1) \) or \( Suf_1(\beta_1) \triangleleft Pref_1(y_1) \). But since \( \beta_1 \) is a handle, \( Suf_1(\beta_1) \triangleright Pref_1(y_1) \). This is again a contradiction.

Hence, we conclude that \( x = y \). If \( A = B \) then obviously \( r = \alpha \). So assume \( A \neq B \) and let \( B \rightarrow \beta_1 \) in \( P \) be used to derive \( \alpha \beta y \). Since \( G \) is SPG so \( \beta_1 \neq \beta \).

Then we have \( S' \xrightarrow{r_m} rBy \xrightarrow{r_m} r_1 y_1 y = \alpha \beta y \). Note \( \beta \neq \varepsilon \) and \( \beta_1 \neq \varepsilon \) both are handles (no \( \varepsilon \)-production allowed). The either \( \beta_1 = \beta \beta \) or \( \beta_1 = \beta_1 \beta, \beta_1 \neq \varepsilon \) and \( \beta \neq \varepsilon \).

Case 1.: \( \beta_1 = \beta \beta, \beta \neq \varepsilon \)

\[ S' \xrightarrow{r_m} rBY \xrightarrow{r_m} r\beta_1 y = \alpha \beta y \sothat r\beta_1 = \alpha \] . Thus \( Suf_1(\beta') = Suf_1(\alpha) \) \( Pref_1(\beta) \). But \( B \rightarrow \beta_1 \beta, \beta_1 \neq \varepsilon \). Hence, \( Suf_1(\beta') \triangleleft Pref_1(\beta), \) a contradiction.

Case 2.: \( \beta = \beta_1 \beta, \beta_1 \neq \varepsilon \)

This can be handled in the same fashion as Case 1. \( S' \xrightarrow{r_m} rBy \xrightarrow{r_m} r\beta_1 \beta y = \alpha \beta y \) and \( r\beta_1 = \alpha \). Thus \( Suf_1(y \beta_1) = Suf_1(\beta_1') = Suf_1(\alpha) \) \( Pref_1(\beta) \). But \( B \rightarrow \beta_1 \varepsilon \).
P. Hence, \( \text{Suf}_1(\beta_1) = \text{Pref}_1(\beta) \), a contradiction.

Thus we conclude that \( A = B \) and \( r = a \) and that every simple precedence grammar is an \( LR(1) \) grammar. But if a \( LR(1) \) grammar has an \( \varepsilon \)-production then it is not a simple precedence grammar.

Example 4:

Let \( G \) be a grammar with the production

\[
\begin{align*}
S & \rightarrow aA \\
A & \rightarrow Ab \mid d 
\end{align*}
\]

All right-sentential forms have the form \( aA^i \), \( i \geq 0 \) and \( A \rightarrow \varepsilon \). This shows that \( G \) is a \( LR \) grammar. However, \( a \notin A \) and \( a \ll A \) both hold making grammar non-simple precedence.

Corollary 1: Every SPG is unambiguous.
CHAPTER IV

WEAK PRECEDENCE GRAMMAR

Definition of Weak Precedence Grammar

Definition 7:

I. Let G be a proper CFG without e-production. Then G is a weak precedence grammar (WPG for short) iff

1. \( \Rightarrow \) is disjoint from \( \ll \) and \( = \),

2. Neither \( X \ll B \) nor \( X = B \) whenever \( A \rightarrow \alpha X\beta \) and \( B \rightarrow \beta \) are in \( P \).

II. In addition if it is uniquely invertible, it is called UIWPG.

Theorem 5: \( \text{SPG} \subset \text{UIWPG} \) \( \bigm/ \{ \text{AU1} \} \text{p.420} \)

Proof:

If G is a SPG then 1 in the definition of WPG is satisfied. For 2 let \( A \rightarrow \alpha X\beta \) and \( B \rightarrow \beta \) be in \( P \). If \( X \ll B \) then \( \exists C \rightarrow \alpha XY\delta \) such that \( Y \rightarrow Br \) and hence \( Y \rightarrow \beta r \). Therefore, \( X \ll \text{Pref}_1(\beta) \). But \( A \rightarrow \alpha X\beta \) implies \( X \not\ll \text{Pref}_1(\beta) \). This is a contradiction.

If \( X = B \) then \( \exists C \rightarrow \alpha XB\delta \). Since \( B \rightarrow \beta \), \( X \ll \text{Pref}_1(\beta) \). But \( A \rightarrow \alpha X\beta \) implies \( X \not\ll \text{Pref}_1(\beta) \) a contradiction.

To see the proper inclusion consider a grammar with productions \( S \rightarrow aA, A \rightarrow Ab | b \). This grammar is UIWPG but not SPG, since \( a \not\ll A, a \ll A, a \not\ll b, \) and \( b \gg b \).
Theorem 6: UIWPG \nsubseteq LR(1)(AU1) p. 424

Proof:

Let G' be the augmented grammar corresponding to the UIWPG. Let S' \xrightarrow{r} Aw \xrightarrow{rm} \alpha \beta \cdot w, S' \xrightarrow{r} rBx \xrightarrow{rm} \alpha \beta \cdot y with FIRST\_1(w) = FIRST\_1(y). If x \neq y then we have two cases:

1. \(x_1 = e \Rightarrow x_1 = y\) and B \xrightarrow{r} \beta \cdot x_1 was used to derive \(\alpha \beta \cdot y\). If \(\beta_1 = e\) then S' \xrightarrow{r} rBx \xrightarrow{rm} \alpha \beta \cdot w so that r = \alpha \beta. Thus Suf\_1(\beta) = Suf\_1(r) \nsubseteq Pref\_1(x_1) = Pref\_1(y) = Pref\_1(w). If \(\beta_1 \neq e\) then S' \xrightarrow{r} rBx \xrightarrow{rm} r_1 \cdot x_1 = r_1 \cdot y = \alpha \beta \cdot y so that \(\alpha \beta = r_1 \cdot y\). Thus Suf\_1(\beta) = Suf\_1(\beta_1) \nsubseteq Pref\_1(x_1) = Pref\_1(y) = Pref\_1(w). Hence Suf\_1(\beta) = Suf\_1(w) or Suf\_1(\beta) \supseteq Pref\_1(w). This is a contradiction.

2. \(y_1 \neq e \Rightarrow x = y_1 \cdot y\) and B \xrightarrow{r} \beta \cdot y was used to derive \(\alpha \beta \cdot y\). Then S' \xrightarrow{r} rBy \xrightarrow{rm} r_1 \cdot y_1 \cdot y = \alpha \beta \cdot y so that \(r_1 \cdot y_1 = \alpha \beta\). Hence by Theorem 1, Suf\_1(\beta_1) \subseteq Pref\_1(y_1) or Suf\_1(\beta_1) \supseteq Pref\_1(y_1). But \(\beta_1\) is a handle, so Suf\_1(\beta_1) \supseteq Pref\_1(y_1), a contradiction.

Hence we conclude that x = y.

If A = B then let the last production used to derive \(\alpha \beta \cdot y\) be B \xrightarrow{r} \beta_1. Since G is uniquely invertible, \(\beta_1 \neq \beta\). Also note that \(\beta_1 \neq e\), \(\beta \neq e\), and S' \xrightarrow{r} rBx \xrightarrow{rm} r_1 \cdot y = \alpha \beta \cdot x so that \(r_1 \cdot \beta_1 = \alpha \beta\).
Case 1: \( \beta_1' = \beta' \beta, \beta' \notin \epsilon \)

Thus we have \( A \to \beta \) and \( B \to \beta' \beta \) in \( P \). By definition of WPG neither Suf\(_1(\beta') \) \( \preceq \) A or Suf\(_1(\beta') \) \( \equiv A \). However, \( S' \xrightarrow{rm} rB \xrightarrow{rm} rB' y = \alpha \beta y \) so that \( rB' = \alpha \). But \( A \wedge \) is a right-sentential form with \( \alpha A \) as a part of handle. Hence by Theorem 1, Suf\(_1(\alpha) \) \( \preceq \) A or Suf\(_1(\alpha) \) \( \equiv A \). This is a contradiction.

Case 2: \( \beta = \beta_1' \beta_1, \beta_1' \notin \epsilon \) with \( B \to \beta_1 \) and \( A \to \beta_1' \beta_1 \) in \( P \)

Again by WPG conditions neither Suf\(_1(\beta_1') \) \( \preceq \) A nor Suf\(_1(\beta_1') \) \( \equiv A \). But \( S' \xrightarrow{rm} rB \xrightarrow{rm} rB_1 y = \alpha \beta y = \alpha \beta_1' \beta_1 y \) so that \( r = \alpha \beta_1' \). Again \( rB \) being a part of handle, Suf\(_1(r) \) and \( B \) are related by either \( \preceq \) or \( \equiv \). But Suf\(_1(r) = \) Suf\(_1(\beta_1') \), giving Suf\(_1(\beta_1') \) \( \equiv B \) or Suf\(_1(\beta_1') \) \( \preceq B \), a contradiction. Hence, we conclude that \( A = B \) and hence \( r = \alpha \).

To see the proper inclusion let \( G \) be a grammar with productions \( S \to aA \mid d, A \to d \). This grammar is LR(1) but not UIWPG.

Corollary 2: Every UIWPG is unambiguous.
CHAPTER V

EXTENDED PRECEDENCE GRAMMAR

It is possible to extend the definition of Wirth-Weber precedence relations to pairs of strings rather than pairs of symbols. We shall give a definition of extended precedence relations that relate strings of \( m \) symbols to strings of \( n \) symbols. Our definition is designed within shift-reduce parsing in mind.

**Definition of Extended Precedence Grammar**

Definition 8:

Let \( G \) be a proper CFG without any \( \epsilon \)-production. Then \((m,n)\)-precedence relations \( \prec , \succeq , \succ , \preceq , \succeq \), and \( \succcurlyeq \) are defined on \((\Sigma^* \cup \{\epsilon\})^m \times (\Sigma^* \cup \{\epsilon\})^n\) as follows for \( \delta \in \Sigma^* \):

1. \( \text{Suf}_m(\emptyset) \prec \gamma \) where \( \gamma = \text{Pref}_n(\delta \omega) \text{ if } \text{Suf}_m(\gamma) \not\subseteq \text{V}_t \) or
   \( \text{FIRST}_n(\delta \omega) \text{ if } \text{Suf}_m(\gamma) \not\subseteq \text{V}_t \)

2. \( \text{Suf}_m(\emptyset) \succeq \gamma \) where \( \gamma = \delta_1 \delta_2 \), \( \delta_1 \not\in \epsilon \), \( \delta_2 \not\in \epsilon \) and \( \beta = \text{Pref}_n(\delta_2 \omega) \text{ if } \text{Suf}_m(\emptyset) \not\subseteq \text{V}_t \)
   or
   \( \text{FIRST}_n(\delta_2 \omega) \text{ if } \text{Suf}_m(\emptyset) \not\subseteq \text{V}_t \)

3. \( \text{Suf}_m(\emptyset) \succ \text{Pref}_n(\omega) \).

23
Definition 9:

1. A proper CFG grammar without any ε-production is called an Extended Precedence Grammar if \( \prec, \equiv, \succ \) are pairwise disjoint.
2. In addition if it is uniquely invertible, it is called UIEPG\(_{m,n}\).

Relation between UIEPG and LR

Theorem 7: UIEPG \( \not\subseteq \) LR \((\text{AU1})\) p. 424

Proof:

Let \( G \) be UIEPG\(_{m,n}\). We show \( G \) is a LR(n). Let \( S' \xrightarrow{\text{rm}} \delta \), with FIRST\(_n\)(w) = FIRST\(_n\)(y). Then we have

1. \( s^m s^n \xrightarrow{\text{rm}} \delta w \) with FIRST\(_n\)(w) = FIRST\(_n\)(y).

2. \( s^m s^n \xrightarrow{\text{rm}} \delta x \) with \( r = s^m r', x = x^n, y = y^n \) with FIRST\(_n\)(w) = FIRST\(_n\)(y). For LR it will suffice to show that \( x = y, A = B, \) and \( \delta = \gamma \).

From 1 we note that Suf\(_n\)(\( \delta \)) \( \succ \) Pref\(_n\)(w) = Pref\(_n\)(y). If \( x = y \) then we have the following cases:

1. \( x \not\in \varepsilon \) \( x = y \) and \( A \xrightarrow{\delta x} \). Therefore \( s^m s^n \xrightarrow{\text{rm}} \delta x \) with \( \delta y \) so that \( \delta = \delta \). If \( \delta \not\in \varepsilon \) then Suf\(_m\)(\( \delta \)) = Suf\(_n\)(\( \delta \)) = Pref\(_n\)(y) a contradiction. If \( \delta \in \varepsilon \) then \( r = \gamma \). Also Suf\(_m\)(\( \gamma \)) = Suf\(_n\)(\( r \)) \( \not\subseteq \) Pref\(_n\)(x\_1\_x) = Pref\(_n\)(y). Again this is a contradiction.

2. \( y \not\in \varepsilon \) \( x = y \) and \( B \xrightarrow{\delta} \). Then \( s^m s^n \xrightarrow{\text{rm}} \delta y \) so that \( r \delta y \) \( \not\subseteq \) Pref\(_n\)(y\_1\_y) = Pref\(_n\)(y\_1\_y) because FIRST\(_n\)(w) = FIRST\(_n\)(y). This is a contradiction.
Hence we conclude that $x = y$.

Now if $A = B$ then let $B \xrightarrow{\delta} \delta_1$ so that $r^m s^n \xrightarrow{rm} rBx \xrightarrow{rm} r\delta_1 x = \gamma \delta x$.

Since $G$ is uniquely invertible, $\delta = \delta_1$. Also, $rBx \xrightarrow{r\delta_1 x} = \delta x$ so that $r \delta_1 = \gamma \delta$.

Hence we have the following cases:

Case 1: $\delta = \delta_1 \delta', \delta_1 \notin \epsilon$

Then $r^m s^n \xrightarrow{rm} rBx \xrightarrow{rm} r\delta' \delta x = \gamma \delta x$ with $r = \gamma \delta'$. Since $\delta' \notin \epsilon$ and $\delta \notin \epsilon$,

$\text{Suf}_m(\gamma) = \text{Suf}_m(\gamma) = \text{Suf}_m(\gamma)$ and $\text{Suf}_m(\gamma) \ll \text{Pref}_n(\gamma)$. Hence $\text{Suf}_m(\gamma) \ll \text{Pref}_n(\delta x)$. But also $\text{Suf}_m(\gamma \delta') \gg \text{Pref}_n(\delta x)$. This is a contradiction.

Case 2: $\delta = \delta_1', \delta_1 \notin \epsilon$

Let $B \xrightarrow{\delta_1}$ be used to derive $\gamma \delta y$. Then $r^m s^n \xrightarrow{rm} rBx \xrightarrow{rm} r\delta_1 x = \gamma \delta x$

so that $r\delta_1 x = \gamma \delta_1' \delta_1 x$ implies $r = \gamma \delta_1'$. Hence, $\text{Suf}_m(r) \ll \text{Pref}_n(\delta_1 x) = \text{Pref}_n(\delta_1 w)$. Also $r^m s^n \xrightarrow{\gamma \delta w} = \gamma \delta_1' \delta_1 w$ implies $\text{Suf}_m(\gamma \delta_1') \ll \text{Pref}_n(\delta_1 w)$ so $\text{Suf}_m(r) \ll \text{Pref}_n(\delta_1 w)$. This is a contradiction.

Hence we conclude that $A = B$ and $r = \gamma$.

To see the proper inclusion we consider a grammar $S \xrightarrow{aA|b} A \xrightarrow{b} b$. This is LR but not UIEPG.

Theorem 8: A grammar is SPG iff it is UIEPG for $m = n = 1$. ([AU1] p. 412, 424, 425)

Proof:

It is obvious.

Theorem 9: UIWPG $\not\subseteq$ UIEPG ([AU1] p. 425)

Proof:

It is trivial.
CHAPTER VI

CONCLUSION

Hierarchical Relations Between Precedence Grammars

Unambiguous CFG

LR

LR(1)

UIWPG

UIEPG

SPG

Language Hierarchies

CFL

= LR(1)

SPL = UIWPL

LL lang

26
Comparison of Precedence and LR Parsing Techniques

(1,1)-precedence grammars are easy to parse. However, removal of precedence conflicts may require addition of many single productions of the form A X. Also not all deterministic CFL's have precedence grammars. On the other hand the canonical set of LR tables can be impractically large for a grammar of practical interest for k \not= 1. Still LR techniques are more suitable than precedence techniques due to the following reasons:

1. LR grammars are the largest natural class of unambiguous grammars from which we can construct parsers.
2. LR(k) parsing is more general than any of the shift-reduce techniques.
3. LR parser size can be cut down by use of table optimization techniques.
4. LR(k) parsing has much better error detecting capabilities as compared to precedence techniques. For example, let Xa be a prefix of some sentence in LR language and Xab not a prefix of any sentence. If we have an input string XabY then canonical parser parses X, shifts a and announces error when b becomes lookahead for the first time. In general LR parser will announces error at the earliest possible opportunity in left-to-right scan of input string. Precedence parser do not enjoy this early error detecting capability. For example, in parsing the same string XabY it is possible for precedence parser to scan arbitrarily many symbols of Y before announcing error.
5. LR parsers are fast.
REFERENCES

Books

{HU} Hopcroft, J. E., and Ullman, J. D. *Formal Languages and Their Relation to Automata*. Massachusetts: Addison-Wesley & Reading, 1969.

Manuals


GLOSSARY

Ambiguous: G is a grammar if there is a sentence w in L(G) with two or more distinct rightmost derivations.

CFG: Is abbreviate of context-free grammar if grammar G for each production P is of the form A → α. Where A is N and α in (N U Σ)*.

Grammar: A grammar is four-tuple G = (N, Σ, P, S) where
1. N is a finite set of nonterminal symbols,
2. Σ is a finite set of terminal symbols,
3. P is a finite subset of (N U Σ)*N(N U Σ)*X(N U Σ)*
4. S is a distinguished symbol in N called the sentence (or start) symbol.

Language: A language over an alphabet is a set of strings over Σ. FORTRAN, PL/I, COBOL and English are language.

Parse: To build a syntactic tree.

Proper Grammar: If in grammar G there is no derivation of the form A → A for any A in N. G is said to be a proper grammar.

Rightmost: If G is a grammar and S = C₀, C₁, C₂, ..., Cₙ. If Cᵢ₊₁ is obtained from Cᵢ by replacing the rightmost nonterminal in Cᵢ by its distinct descendants, then the associated derivation C₀, C₁, ..., Cₙ is called rightmost derivation of Cₙ from C₀ in G.
FIRST\(_k(\alpha)\) for a grammar \(G = (N, \Sigma, P, S)\), \(\text{FIRST}_k(\alpha) = (x \mid \alpha \xrightarrow{\star \text{rm}} x\beta \text{ and } |x| = k)\). We say that \(\text{FIRST}_k(\alpha)\) consists of all terminal prefixes of length \(k\) of the terminal strings that can be derived from \(\alpha\).

All small alphabet \(a, b, c\) represent symbols.

The letters \(w, x, y\) represent strings.

The Greek letters \(\alpha, \beta, \gamma, \delta, \xi, \zeta\) represent the strings of symbols.

The capital alphabet \(A, B, X, Y\) represent nonterminal symbols.

\[\xrightarrow{\rightarrow} \]

If \((\alpha, \beta)\) is a production, we use the descriptive shorthand \(\alpha \xrightarrow{\rightarrow} \beta\).

To be read as directly derives.

\[\xrightarrow{+} \]

To be read derives in a nontrivial way iff \(\alpha \xrightarrow{i} \beta\) for some \(i \geq 1\).

\[\xrightarrow{*} \]

To be read derived iff \(\alpha \xrightarrow{i} \beta\) for some \(i \geq 0\).

\[\xrightarrow{\text{rm}} \]

If \(S \xrightarrow{\text{rm}} \alpha_0, \alpha_1, ..., \alpha_n\) is a rightmost derivation in grammar \(G\), then we shall write \(S \xrightarrow{\star \text{Grin}} \alpha_n\) or \(S \xrightarrow{\star \text{rm}} \alpha_n\), and call \(\alpha_n\) is a right sentential form.