algorithms

MDPI

Article

Precedence Table Construction Algorithm for CFGs Regardless of Being OPGs

Leonardo Lizcano, Eduardo Angulo and José Márquez *

Department of Computer Science and Engineering, Universidad del Norte, Barranquilla 081007, Colombia; ldlizcano@uninorte.edu.co (L.L.); edangulo@uninorte.edu.co (E.A.)
* Correspondence: jmarquez@uninorte.edu.co

Abstract: Operator precedence grammars (OPG) are context-free grammars (CFG) that are characterized by the absence of two adjacent non-terminal symbols in the body of each production (right-hand side). Operator precedence languages (OPL) are deterministic and context-free. Three possible precedence relations between pairs of terminal symbols are established for these languages. Many CFGs are not OPGs because the operator precedence cannot be applied to them as they do not comply with the basic rule. To solve this problem, we have conducted a thorough redefinition of the Left and Right sets of terminals that are the basis for calculating the precedence relations, and we have defined a new Leftmost set. The algorithms for calculating them are also described in detail. Our work's most significant contribution is that we establish precedence relationships between terminals by overcoming the basic rule of not having two consecutive non-terminals using an algorithm that allows building the operator precedence table for a CFG regardless of whether it is an OPG. The paper shows the complexities of the proposed algorithms and possible exceptions to the proposed rules. We present examples by using an OPG and two non-OPGs to illustrate the operation of the proposed algorithms. With these, the operator precedence table is built, and bottom-up parsing is carried out correctly.

Keywords: operator precedence language; operator precedence grammar; precedence table; precedence relations; algorithm

Citation: Lizcano, L.; Angulo, E.; Márquez, J. Precedence Table Construction Algorithm for CFGs Regardless of Being OPGs. *Algorithms* **2024**, *17*, 345. https://doi.org/10.3390/a17080345

Academic Editor: Dimitris Fotakis

Received: 24 June 2024
Revised: 29 July 2024
Accepted: 30 July 2024
Published: 7 August 2024

1. Introduction

This paper delves into the operator precedence languages family (OPL), created by R. Floyd [1], which is used for efficient bottom-up parsing [2,3]. The author's inspiration from the structure of arithmetic expressions, where precedence is assigned to multiplicative operators ($*$ and $/$) over additive operators ($+$ and $-$), is a key insight. Through the process of bottom-up parsing, the left side (non-terminal symbol) completely replaces the identified right-side production (body) in a context-free grammar (CFG), leaving no room for ambiguity. Operator precedence was extended to grammars that consist of terminals on the right side, even if non-terminals separated them. A CFG is generally considered an operator precedence grammar (OPG) if it does not contain productions with adjacent non-terminal symbols on its right-hand side.

OPLs have greatly aided in inferring context-free grammars. An example is described in [4–6], where a method is presented for inferring operator precedence-free grammars from bracketed samples. The method is based on constructing parse trees and utilizing functions that produce the grammars. Precedence rules are used to identify the brackets surrounding the terminal elements of a string in the sample, indicating the order in which the string should be evaluated.

Research on input-driven formal languages (IDL) [7–9] (later renamed as Visibly Pushdown Languages (VPL) [10]) has concluded [11] that OP closure properties imply ID closure properties and that ID languages are a specific type of OP languages characterized by limited OP relations [12].

Since the advancements in parallel computing, local parsability properties derived from OPL have been utilized to generate fast parallel parsers [13].

This paper's main contribution is its novel approach to building precedence relationships between terminal symbols in non-OPGs. This approach allows the generation of operator precedence tables and bottom-up parsing on a string derived by the non-OPG with support from the table obtained. The novelty of the work is a significant aspect that sets it apart from previous research in the field.

- The sets of Left and Right terminals proposed in previous works [1,11,14] to build precedence relationships between the terminals of an OPG were redefined.
- A new set of terminals called Leftmost was defined to support solving precedence relationships when there are two or more adjacent terminals on the right side of the productions. This set is significant as it systematically handles the precedence relationships in such complex scenarios, expanding the bottom-up parsing process with non-OPGs.
- An algorithm was established to build the precedence table with the support of the three sets of terminals proposed.
- Finally, applying the proposed algorithms makes it possible to obtain from a non-OPG the relations and operator precedence table previously limited to OPGs only. In addition, a bottom-up operator-precedence parser could be used to parse any string generated by a non-OPG, a technique that could only be applied to OPGs.

As demonstrated in this work, the proposed solution to construct the operator precedence table is not just a theoretical concept but a practical tool that can be applied to OPGs without any restrictions. At the end of the work, the functionality is demonstrated by applying it to an OPG and two non-OPGs and carrying out the respective bottom-up parsing. This practical demonstration should instill confidence in the effectiveness of the proposed solution and ensure its practical applicability.

The paper is organized as follows: Section 2 provides the necessary definitions to understand this work. Section 3 presents the previous work carried out in the area of CFG. Section 4 introduces our approach, detailing the problem we aim to solve and the redefinitions of previous works' concepts used to develop our algorithms. Section 5 presents examples of applying the proposed algorithm to one OPG and two non-OPGs, demonstrating the application of bottom-up parsing with the precedence tables to test their use. Section 6 describes several exceptions where the rules described in the proposed algorithms do not apply. Finally, Section 7 provides conclusions to our work.

2. Basic Definitions

2.1. Context-Free Grammars and Languages

Within Chomsky's hierarchy [15], context-free grammars play a significant role in programming and compilation language applications by addressing the syntactic structure of programming languages.

A *context-free grammar* (CFG) G generates a language $L(G)$, commonly called *context-free language* (CFL), which is composed of the strings generated by the CFG. The CFG is defined as a 4-tuple $G(S) =< T, N, S, P >$, where:

- T represents the set of terminal symbols that create the strings within the CFL generated by the CFG.
- N represents the set of non-terminal symbols or syntactic variables that determine strings leading to CFL generation.
- S is the initial non-terminal symbol from which all strings defined by the GFC are generated.
- P is the set of productions or rules that group terminals and non-terminals to generate the strings. A production takes the form $A \longrightarrow \alpha$, where A represents a non-terminal and α is a combination of terminals and non-terminals. The $\longrightarrow$ is pronounced "produce". Thus, the production is read "A produce alpha".

The following notation will be used: lowercase letters from the beginning of the alphabet represent terminal symbols $(a, b, c, \ldots \in T)$; uppercase letters from the beginning of the alphabet represent non-terminal symbols $(A, B, C, \ldots \in N)$; lowercase letters late in the alphabet represent terminal strings $(u, v, w, x, y, z \in T^+)$; lowercase Greek letters generally represent terminal and/or non-terminal strings $(\alpha, \beta, \gamma, \ldots \in (T \cup N)^*)$, and the empty string will be represented by ε (this Greek letter will be the exception to the above convention).

2.2. Operator Precedence Grammars (Basic Rule)

A CFG is considered an operator grammar if none of the productions in P have adjacent non-terminal symbols on their right side. In other words, for a production $A \longrightarrow \alpha$ in G, α does not take the form $uBCv$.

2.3. Derivations

A string $\eta \in (T \cup N)^*$ can be derived in k steps from a non-terminal $A \in N$ if there exists a sequence of strings of the form $\eta_1, \eta_2, \ldots, \eta_k$. Using the symbol $\Longrightarrow$, meaning *one-step derivation*, the sequence of strings is arranged in the form: $\eta_1 \Longrightarrow \eta_2 \Longrightarrow \ldots \Longrightarrow \eta_k$.

In this sequence, η_1 corresponds to A and η_k to η. Each intermediate η_i in the sequence has the form $\delta B \gamma$, where $B \in N$ and there is a production $B \longrightarrow \beta$, such that by substituting it on the right side, $\eta_{i+1} (= \delta \beta \gamma)$ is obtained. The *derivation in one or more steps* is denoted with the operator $\overset{+}{\Longrightarrow}$. In general, it is stated that every string of terminal symbols $w \in L(G)$ can be established as $S \overset{+}{\Longrightarrow} w$ or expanding the derivation in one or more steps: $S = \eta_1 \Longrightarrow \eta_2 \Longrightarrow \ldots \Longrightarrow \eta_k = w$. For a grammar G, a sentential form $\eta \in (T \cup N)^*$ is a string that $S \overset{*}{\Longrightarrow} \eta$, meaning that η can be obtained with zero or more derivations from S. If two grammars generate identical languages, they are considered *equivalent*.

2.4. Types of Derivations

According to [16], $S = \eta_1 \Longrightarrow \eta_2 \Longrightarrow \ldots \Longrightarrow \eta_k = w$ is a leftmost derivation of the string $w \in T^*$, when each $\eta_i, 2 \leq i \leq k-1$, has the form $x_i A_i \beta_i$ with $x_i \in T^*$, $A_i \in N$ and $\beta_i \in (T \cup N)^*$. Furthermore, $A_i \longrightarrow \alpha_i$ belongs to P, and then, in each η_i, the symbol A_i is substituted by α_i, resulting in the sentential form η_{i+1}. Each η_i is called the left-sentential form.

If each $\eta_i, 2 \leq i \leq k-1$, takes the form of $\beta_i A_i x_i$ with $x_i \in T^*$, $A_i \in N$, and $\beta_i \in (T \cup N)^*$, then a rightmost derivation would be formed. Each η_i is called the right-sentential form.

2.5. Parse Trees

According to reference [16], a derivation tree for a CFG $G(S) =< T, N, S, P >$ is a labeled and ordered tree in which each node receives a label corresponding to a symbol from the set $N \cup T \cup \{\varepsilon\}$. If a non-leaf node is labeled A and its immediate descendants are labeled $X_1, X_2, \ldots, X_n$, then $A \longrightarrow X_1 X_2 \ldots X_n$ is a production in P. A labeled ordered tree D is a derivation tree for a CFG $G(A) =< T, N, A, P >$ if

1. The root of D is labeled A.
2. If $D_1, \ldots, D_k$ are the subtrees of the direct descendants of the root and the root of D_i is labeled X_i, then $A \longrightarrow X_1 \ldots X_k$ is in P.
3. D_i is a derivation tree for $G(X_i) =< T, N, X_i, P >$ if X_i is in N or D_i is a single node labeled X_i if X_i is in T or D_i is a single node labeled ε.

Parse trees can be constructed from any derivation type.

2.6. Bottom-Up Parsing

The bottom-up parsing syntax analysis [2], also known as shift-reduce parsing, attempts to build a parsing tree for an input string that starts in the leaves (tree bottom) and moves to the root (tree top). This process is considered as the reduction of the string w to the initial symbol S of a CFG. At each step of parsing reduction, a particular substring of the sentential form that matches the right side of a production is replaced by the non-terminal symbol of the left side of that production. If, at each step, the substring is chosen correctly, a rightmost derivation is traced out inversely.

2.7. Handle

A handle [2,17] of a right-sentential form γ is a production $A \longrightarrow \beta$ and a position of γ where the string β could be found and replaced by A to produce the previous right-sentential form in a rightmost derivation of γ. Generally, if $S \overset{*}{\Longrightarrow} \alpha Aw \Longrightarrow \alpha\beta w$, then $A \longrightarrow \beta$ if the position next to α is a handle of $\alpha\beta w$. If the grammar is ambiguous, more than one handle will be obtained. If the grammar is unambiguous, then every right-sentential form has exactly one handle. Usually, the string β is told to be a handle of $\alpha\beta w$ when the conditions for doing so are clear.

2.8. Implementation of a Bottom-Up Parsing

The bottom-up parsing by shift and reduction uses a stack to store grammar symbols and a buffer to handle the input string to be analyzed. The $ symbol is also used to delimit the bottom of the stack and the right side of the input buffer. The recognition model format starts with the stack at $ and the string w in the input, as shown in Table 1.

Table 1. Bottom-up parsing table.

Stack	Input	Action
$	w	
...	...	Shift/Reduce
...	...	Shift/Reduce
$S	$	***Accept***

The bottom-up parsing shifts zero or more input symbols to the stack until a handle is at the top. The parsing then reduces the handle to the left side of the appropriate production to obtain the sentential form corresponding to the previous step of the rightmost derivation. When reducing, the parsing recognizes that the right end of the handle is at the top of the stack. Therefore, the parsing takes action to find its left end in the stack and determine the non-terminal that will replace it according to the right side of some production where it matches the handle. These steps are repeated until an error is detected or until the stack contains $$S$$ and the input is with the symbol $, at which point the parsing ends and the input string is considered valid for the CFG.

3. Previous Work

According to [1,11,14], for a CFG G, the left terminal sets $\mathcal{L}_G(A)$ and right terminal sets $\mathcal{R}_G(A)$ are defined as follows:

$$\mathcal{L}_G(A) = \left\{ a | A \overset{*}{\Longrightarrow} \gamma a \alpha, \gamma \in N \cup \{\varepsilon\} \right\}.$$

$$\mathcal{R}_G(A) = \left\{ a | A \overset{*}{\Longrightarrow} \alpha a \gamma, \gamma \in N \cup \{\varepsilon\} \right\}.$$

Precedence relationships arise from the parse tree's establishment of binary relationships between consecutive terminals or those that become sequential after a bottom-up process toward the non-terminal symbol S.

The precedence relationships are established as follows:

1. $a \doteq b$, if and only if $\exists A \longrightarrow \alpha a \gamma b \beta \in P, \gamma \in N \cup \{\varepsilon\}$.
2. $a \lessdot b$, if and only if $\exists A \longrightarrow \alpha a D \beta \in P, b \in \mathcal{L}_G(D)$.
3. $a \gtrdot b$, if and only if $\exists A \longrightarrow \alpha D b \beta \in P, a \in \mathcal{R}_G(D)$.

According to [2], precedence relations are used to delimit a handle in a right-sentential form. That is, in the sentential form $\alpha\beta w$, the substring β is a handle if there is a production $A \longrightarrow \beta$ in the CFG; therefore, it can be delimited by the precedence relations, obtaining $\alpha \lessdot \beta \gtrdot w$. The relation $\doteq$ between the symbols used when the handle has more than one consecutive terminal symbol. That is, if $\beta = a_1 a_2 \dots a_n$, then the relation $a_i \doteq a_{i+1}, \forall i = 1, \dots, n-1$ should be established. Suppose a right-sentential form is $A_0 a_1 A_1 a_2 \dots a_n A_n$, where each A_i is a non-terminal. In that case, the relations can be established as $a_i \lessdot A_i a_{i+1}$, with a maximum of one non-terminal (operator grammar principle) between the terminals. Non-terminals could be eliminated in the right sentential form, leaving only the relationships between terminals.

In short, each time a handle is obtained, it can be enclosed between the symbols of $\lessdot$ and $\gtrdot$. In addition, two consecutive terminal symbols can be inside a handle (non-terminals in the middle), and the relationship is to be established $\doteq$. Finally, non-terminal symbols can be removed from the right-sentential form when a handle is found and locked between the precedence relations.

According to [2], the concepts discussed in Section 2.8, and the precedence relations, the precedence parsing Algorithm 1 is constructed as follows:

Algorithm 1 Precedence parsing algorithm

```
Require: Set of precedence relations.
  procedure PrecedenceParsing(w)
      stack ← [$]
      end ← false
      pos ← 0
      do
          a ← stack[top]
          b ← w$[pos]
          if a = $ and b = $ then
              print("Action: Accept")
              end ← true
          else
              if a ⋖ b or a ≐ b then
                  pos + +
                  push(stack, b)
                  print("Action: Shift")
              else
                  if a ⋗ b then
                      do
                          e ← pull(stack)
                      while not (stack[top] ⋖ e)
                      print("Action: Reduce")
                  else
                      print("Action: Error")
                      end ← true
                  end if
              end if
          end if
      while not end
  end procedure
```

4. Approach

This work proposes a redefinition of the *Left* and *Right* terminal sets, corresponding to $\mathcal{L}_G(A)$ and $\mathcal{R}_G(A)$, respectively. Additionally, the definition of the new *Leftmost* set is presented. The new rules for determining precedence relations are demonstrated even if the CFG does not comply with the fundamental rule of operator grammar, meaning that the CFG can have more than one adjacent non-terminal symbol on the right side of its productions.

We present the algorithms for constructing the three proposed sets: *Left*, *Leftmost*, and *Right*. Additionally, we demonstrate the algorithm that incorporates the newly established precedence rules.

4.1. Definitions

There are the following symbols:

- A string of terminals $x \in T^+$.
- A string of non-terminals $\beta \in N^*$.
- A sentential form $\gamma \in (T \cup N)^*$.

The following sets are defined:

- The left, denoted as $\mathcal{L}(A)$, is the set of terminal symbols that can appear on the left side of any sentential form as a product of a rightmost derivation from A.

$$\mathcal{L}(A) = \{a \in T \mid A \overset{*}{\Longrightarrow} \beta a \gamma\}.$$

- The leftmost, denoted as $\mathcal{L}_m(A)$, is the set of terminal symbols that can appear at the end of the left side in any derivation from A, in the absence of other leftmost grammatical symbols (or that derive to ε in their absence).

$$\mathcal{L}_m(A) = \{a \in T \mid A \overset{*}{\Longrightarrow} a\gamma\}.$$

- The right, denoted as $\mathcal{R}(A)$, is the set of terminal symbols that can appear at the right end of the rightmost derivations from A or appear further to the right in any production of A.

$$\mathcal{R}(A) = \{a \in T \mid A \overset{*}{\Longrightarrow} \gamma a \vee A \longrightarrow \gamma a \beta\}.$$

According to the above, the following precedence rules are defined:

- If $C \longrightarrow \gamma_1 a \beta B \gamma_2$, $b \in \mathcal{L}(B)$, then $a \lessdot b$.
- If $C \longrightarrow \gamma_1 A \beta B \gamma_2$, $a \in \mathcal{R}(A) \wedge b \in \mathcal{L}_m(B) \wedge \beta \overset{*}{\Longrightarrow} \varepsilon$, then $a \gtrdot b$.
- If $C \longrightarrow \gamma_1 A \beta b \gamma_2$, $a \in \mathcal{R}(A) \wedge \beta \overset{*}{\Longrightarrow} \varepsilon$, then $a \gtrdot b$.
- If $C \longrightarrow \gamma_1 a \beta b \gamma_2$, then $a \doteq b$.

4.2. Problem Analysis

4.2.1. Absolute Operator Grammar

Given the rightmost derivation series of the form:

$$S \overset{+}{\Longrightarrow} \gamma_1 A x_1 \Longrightarrow \gamma_1 \omega_1 x_1, \text{ where } A \longrightarrow \omega_1 \text{ and } \omega_1 = \gamma_2 B \gamma_3. \tag{1}$$

It is noted that ω_1 is a handle according to the previous definition in Section 2.7.
The right-sentential form of (1) can be rewritten as follows:

$$\gamma_1 \omega_1 x_1 = \gamma_1 \gamma_2 B \gamma_3 x_1 \overset{+}{\Longrightarrow} \gamma_1 \gamma_2 B x_2 x_1 \Longrightarrow \gamma_1 \gamma_2 \omega_2 x_2 x_1, \text{ where } B \longrightarrow \omega_2. \tag{2}$$

From the last sentential form in (2), it follows that ω_2 is a handle according to the definition in Section 2.7.

Because of the above, the analysis will focus on the productions born of A, which have at least one non-terminal B on the right, as seen in (1). The productions in A will be traversed through grammar to find the relations of precedence of non-terminal B located on the right side.

Within the context of ω_1, having to $B \longrightarrow \omega_2$, and ω_2 is a handle, the following forms can be obtained from its definition in (1):

(a) $\omega_1 = \gamma_4 aB$, it follows that $a \lessdot \mathcal{L}(B)$. Here $\gamma_2 = \gamma_4 a$ and $\gamma_3 = \varepsilon$.
(b) $\omega_1 = B b \gamma_5$, it follows that $\mathcal{R}(B) \gtrdot b$. Here $\gamma_2 = \varepsilon$ and $\gamma_3 = b\gamma_5$.
(c) $\omega_1 = \gamma_4 aBb\gamma_5$, it follows that $a \lessdot \mathcal{L}(B)$ and $\mathcal{R}(B) \gtrdot b$. Here $\gamma_2 = \gamma_4 a$ and $\gamma_3 = b\gamma_5$.

For the non-terminal start symbol, S, an initial dummy production is defined $E \longrightarrow \$S\$$ and rule (c) is applied.

For a production $A \longrightarrow \gamma_1 aXb\gamma_2, X \in N \cup \{\varepsilon\}$, it follows that $a \doteq b$.

4.2.2. General Grammar with Two or More Consecutive Non-Terminal Symbols

Redefinition of Left, Right and definition of Leftmost.

Left(B):

Given the rightmost derivation:

$$\gamma_1 aBx_1 \Longrightarrow \gamma_1 a\omega_1 x_1. \tag{3}$$

It is observed that ω_1 is a handle within a bottom-up parsing (inverse rightmost derivation) since $B \Longrightarrow \omega_1$; therefore, from (3), it can be established that $\gamma_1 a \lessdot \omega_1 \gtrdot x_1$. In addition, ω_1 can have the following forms:

(a) $\omega_1 = \beta_1 b\gamma_2$.
(b) $\omega_1 = \beta_1 C\gamma_3$.

Continuing with the rightmost derivation in (3) and taking each of the options of ω_1, we obtain that:

(a) $\gamma_1 aBx_1 \Longrightarrow \gamma_1 a\beta_1 b\gamma_2 x_1$ since ω_1 is a handle, consequently $\gamma_1 a \lessdot \beta_1 b\gamma_2 \gtrdot x_1$; therefore, $a \lessdot b$ where $B \longrightarrow \beta_1 b\gamma_2$ and $b \in \mathcal{L}(B)$.
(b) $\gamma_1 aBx_1 \Longrightarrow \gamma_1 a\beta_1 C\gamma_3 x_1 \overset{*}{\Longrightarrow} \gamma_1 a\beta_1 Cx_2 x_1 \overset{*}{\Longrightarrow} \gamma_1 a\beta_1\beta_2 Dx_3 x_2 x_1 \Longrightarrow \gamma_1 a\beta_1\beta_2\beta_3 b\gamma_4 x_3 x_2 x_1$. It is observed that $C \overset{+}{\Longrightarrow} \beta_2 Dx_3$ and $D \longrightarrow \beta_3 b\gamma_4$; therefore, $\beta_3 b\gamma_4$ is a handle, and consequently $\gamma_1 a\beta_1\beta_2 \lessdot \beta_3 b\gamma_4 \gtrdot x_3 x_2 x_1$ and $a \lessdot b$.
In short, $\gamma_1 aBx_1 \overset{+}{\Longrightarrow} \gamma_1 a\beta_1\beta_2\beta_3 b\gamma_4 x_3 x_2 x_1$; taking $\beta = \beta_1\beta_2\beta_3$ and $\gamma_5 = \gamma_4 x_3 x_2$, we obtain the summary derivation $\gamma_1 aBx_1 \overset{+}{\Longrightarrow} \gamma_1 a\beta b\gamma_5 x_1$, where $B \overset{+}{\Longrightarrow} \beta b\gamma_5$ and $b \in \mathcal{L}(B)$.

Right(B):

Given the rightmost derivation:

$$\gamma_1 Bax_1 \Longrightarrow \gamma_1 \omega_1 ax_1. \tag{4}$$

It is observed that ω_1 is a handle within a bottom-up parsing (inverse rightmost derivation), since $B \Longrightarrow \omega_1$; therefore, from (4), it can be established that $\gamma_1 \lessdot \omega_1 \gtrdot ax_1$. In addition, ω_1 can have the following forms:

(a) $\omega_1 = \gamma_2 b\beta$.
(b) $\omega_1 = \gamma_3 C\beta_1, \beta_1 \overset{*}{\Longrightarrow} \varepsilon$.

Continuing with the rightmost derivation in (4) and taking each of the options of ω_1, we obtain that:

(a) $\gamma_1 Bax_1 \Longrightarrow \gamma_1\gamma_2 b\beta ax_1$ since ω_1 is a handle, consequently $\gamma_1 \lessdot \gamma_2 b\beta \gtrdot ax_1$; therefore, $b \gtrdot a$ where $B \longrightarrow \gamma_2 b\beta$ and $b \in \mathcal{R}(B)$.
(b) $\gamma_1 Bax_1 \Longrightarrow \gamma_1\gamma_3 C\beta_1 ax_1 \overset{*}{\Longrightarrow} \gamma_1\gamma_3\gamma_4 Dax_1 \Longrightarrow \gamma_1\gamma_3\gamma_4\gamma_5 b\beta_2 ax_1$. It is noted that $C \overset{+}{\Longrightarrow} \gamma_4 D$ and $D \longrightarrow \gamma_5 b\beta_2$; therefore, $\gamma_5 b\beta_2$ is a handle, and consequently,

$\gamma_1\gamma_3\gamma_4 \lessdot \gamma_5 b\beta_2 \gtrdot ax_1$ and $b \gtrdot a$.

In short, $\gamma_1 Bax_1 \overset{+}{\Longrightarrow} \gamma_1\gamma_3\gamma_4\gamma_5 b\beta_2 ax_1$; taking $\gamma_6 = \gamma_3\gamma_4\gamma_5$, we obtain the summary derivation $\gamma_1 Bax_1 \overset{+}{\Longrightarrow} \gamma_1\gamma_6 b\beta_2 ax_1$, where $B \overset{+}{\Longrightarrow} \gamma_6 b\beta_2$ and $b \in \mathcal{R}(B)$.

Leftmost(B):

Given the production with two non-terminal symbols on the right (body):

$$S \longrightarrow AB \tag{5}$$

$$A \longrightarrow \gamma_1 b_1 \beta_1 \tag{6}$$

Rightmost derivation from S in (5) is:

$$\underset{(i)}{S} \Longrightarrow \underset{(ii)}{AB} \overset{+}{\Longrightarrow} \underset{(iii)}{Aa\gamma} \overset{+}{\Longrightarrow} \underset{(iv)}{Aax_1} \Longrightarrow \underset{(v)}{\gamma_1 b_1 \beta_1 ax_1} = \underset{(vi)}{\gamma_2 Cax_1} \overset{+}{\Longrightarrow} \underset{(vii)}{\gamma_2\gamma_3\gamma_4 b_2\beta_2 ax_1}.$$

In summary:

- In (v), the relationship $\lessdot \gamma_1 b_1 \beta_1 \gtrdot ax_1$ can be established since the right part of the production (6) is a handle. Therefore, $b_1 \gtrdot a$ and $b_1 \in \mathcal{R}(A)$.
- In (vi), it is known that $\beta_1 = C_0 C_1 \dots C_n C$, where each $C_i \in N$ and C is the non-terminal, first derived from the right. It is noted that $\gamma_2 = \gamma_1 b_1 C_0 \dots C_n$.
- In (vii), $C \overset{+}{\Longrightarrow} \gamma_3\gamma_4 b_2\beta_2$ being $\gamma_4 b_2\beta_2$ the result of the last rightmost derivation performed and, in turn, a handle; therefore, the relationship $\gamma_2\gamma_3 \lessdot \gamma_4 b_2\beta_2 \gtrdot ax_1$ is established, since $\gamma_4 b_2\beta_2$ is a handle, therefore, $b_2 \gtrdot a$.
- From (iv), $Aax_1 \overset{+}{\Longrightarrow} \gamma_2\gamma_3\gamma_4 b_2\beta_2 ax_1$. Establishing $\gamma_5 = \gamma_2\gamma_3\gamma_4$, then $A \overset{+}{\Longrightarrow} \gamma_5 b_2\beta_2$, therefore, $b_2 \in \mathcal{R}(A)$.
- From (ii), $B \overset{+}{\Longrightarrow} a\gamma$ is established, from which we define the new Leftmost set of B as $\mathcal{L}_m(B) = \{a \in T \mid B \overset{+}{\Longrightarrow} a\gamma\}$.

4.2.3. An Intuitive Way to Observe the Elements of Each Set

Vision of Left($\cdot$):

Given the CFG:

$A \longrightarrow CDE$

$E \longrightarrow eh$

$D \longrightarrow d$

$C \longrightarrow cF$

$F \longrightarrow f$

The rightmost derivation of the string $cfdeh$ carries the following way of creating the $Left(A)$ or $\mathcal{L}(A)$.

$$\underset{\lessdot}{A} \Longrightarrow \underset{e}{CDE} \Longrightarrow \underset{d}{CD\mathbf{e}h} \Longrightarrow \underset{c}{C\mathbf{d}eh} \Longrightarrow \mathbf{c}Fdeh \Longrightarrow c\mathbf{f}deh.$$

The terminal symbols e, d, and c are viewed from the left and recorded in $\mathcal{L}(A)$. The symbol e is the first one seen from the left and is recorded in $\mathcal{L}(A)$, and the symbol h is blocked by e since both appear when E is substituted. Then, D is replaced by d, which is observed and recorded in $\mathcal{L}(A)$. When cF replaces C, we observe and record c in $\mathcal{L}(A)$. Finally, when the substitution of F by f is carried out, it can no longer be observed (blocked by c) and cannot be registered. Consequently:

$$\mathcal{L}(A) = \{e, d, c\}.$$

Vision of Right($\cdot$):

Given the CFG:

$A \longrightarrow abBC$

$B \longrightarrow eF$

$C \longrightarrow cD$
$D \longrightarrow d$
$F \longrightarrow f$

The rightmost derivation of the string $abefcd$ carries the following way of creating the $Right(A)$ or $\mathcal{R}(A)$.

$$A \underset{b}{\Longrightarrow} a\mathbf{b}BC \underset{c}{\Longrightarrow} abB\mathbf{c}D \underset{d}{\Longrightarrow} abBc\mathbf{d} \Longrightarrow ab\mathbf{e}Fcd \underset{\gg}{\Longrightarrow} abe\mathbf{f}cd.$$

The terminal symbols b, c, and d are viewed from the right and recorded in $\mathcal{R}(A)$. The symbol b is the first one observed and recorded in $\mathcal{R}(A)$. The symbol a is blocked by b, since both appear when A is derived. C is then replaced by cD and c is observed and recorded in $\mathcal{R}(A)$. When D is substituted for d, we observe and record d in $\mathcal{R}(A)$. Then B is derived by eF but e cannot be recorded since the previous inclusion of c and d in $\mathcal{R}(A)$ makes it invisible or blocks it from being observed from the right. Finally, when substituting F for f, it cannot be observed from the right either (blocked by c and d) and cannot be recorded. Consequently:

$$\mathcal{R}(A) = \{b, c, d\}.$$

Vision of Leftmost(·):
Given the CFG:
$S \longrightarrow dAB$
$A \longrightarrow fa$
$B \longrightarrow Cb$
$C \longrightarrow c$

The rightmost derivation of the string $dfacb$ carries the following way of creating the $Leftmost(B)$ or $\mathcal{L}_m(B)$, as shown in Figure 1.

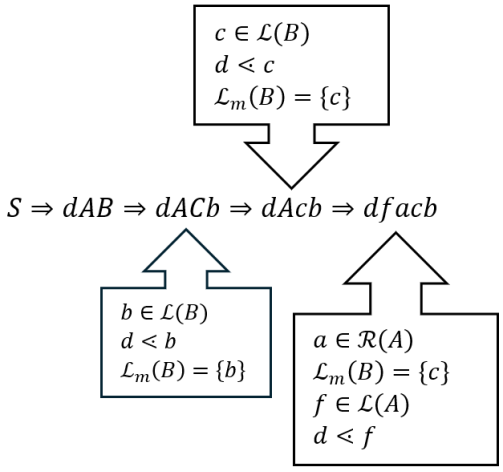

Figure 1. Derivation of $dfacb$ chain and $Leftmost(B)$ set creation sequence.

It can be seen that $\mathcal{L}(B) = \{b, c\}$. When B derivates Cb, we observe that $\mathcal{L}_m(B) = \{b\}$ because no more terminal symbols are seen up to that point. In the following derivation, C is replaced by c; therefore, c is located further to the left of the substring cb, leaving $\mathcal{L}_m(B) = \{c\}$.

4.2.4. Summary of Relations between Grammatical Symbols from Productions and Sets

Table 2 shows the relationships between the grammatical symbols of the productions and the sets of terminals.

Table 2. General productions and precedence relationships obtained.

Producing	Relationship
$A \longrightarrow a\beta b$	$a \doteq b$
$A \longrightarrow C\beta b, \beta \stackrel{*}{\Longrightarrow} \varepsilon$	$\mathcal{R}(C) \gtrdot b$
$A \longrightarrow a\beta C$	$a \lessdot \mathcal{L}(C)$
$A \longrightarrow B\beta C, \beta \stackrel{*}{\Longrightarrow} \varepsilon$	$\mathcal{R}(B) \gtrdot \mathcal{L}_m(C)$

4.3. Algorithms

The following algorithms were developed based on the definitions presented in Section 4.1.

Algorithm 2 receives a non-terminal A whose productions are known; for each of these, it is verified whether they correspond to the form $\beta B\gamma$; if so, Algorithm 2 is executed again, sending the non-terminal B as a parameter (provided that $B \neq A$), and its response is included in the set $\mathcal{L}(A)$.

Subsequently, it is verified whether the production corresponds to the form $\beta a\gamma$, and if so, the terminal a is added to the set $\mathcal{L}(A)$.

Once the verification of all productions of A is completed, the set $\mathcal{L}(A)$ is returned as the output of Algorithm 2.

Algorithm 2 `Left` algorithm of a non-terminal A

```
function LEFT(A)
    L(A) ← ∅
    for each production of A do
        if A ⟶ βBγ then
            L(A) ⊇ Left(B) // L(A) contains a Left(B)
        end if
        if A ⟶ βaγ then
            a ∈ L(A) // a is in L(A)
        end if
    end for
    return L(A)
end function
```

Algorithm 3 receives a non-terminal A whose productions are known; for each of these, it is verified if they correspond to the form $\beta B\gamma$ and $\beta \stackrel{*}{\Longrightarrow} \varepsilon$; if so, Algorithm 3 is executed again, sending the non-terminal B as a parameter (provided that $B \neq A$), and its response is included in the set $\mathcal{L}_m(A)$.

Subsequently, it is verified if the production corresponds to the form $\beta a\gamma$ and $\beta \stackrel{*}{\Longrightarrow} \varepsilon$. If so, the terminal a is added to the set $\mathcal{L}_m(A)$.

Once the verification of all productions of A is completed, the set $\mathcal{L}_m(A)$ is returned as the output of Algorithm 3.

Algorithm 3 `Leftmost` algorithm of a non-terminal A

```
function LEFTMOST(A)
    L_m(A) ← ∅
    for each production of A do
        if A ⟶ βBγ and β ⟹* ε then
            L_m(A) ⊇ Leftmost(B)
        end if
        if A ⟶ βaγ and β ⟹* ε then
            a ∈ L_m(A)
        end if
    end for
    return L_m(A)
end function
```

Algorithm 4 receives a non-terminal A whose productions are known; for each of these, it is verified if they correspond to the form $\gamma B \beta$ and $\beta \overset{*}{\Longrightarrow} \varepsilon$; if so, Algorithm 4 is executed again, sending the non-terminal B as a parameter (provided that $B \neq A$) and its response is included in the set $\mathcal{R}(A)$.

Subsequently, it is verified whether the production corresponds to the form $\gamma a \beta$, and if so, the terminal a is added to the set $\mathcal{R}(A)$.

Once all productions of A have been verified, the set $\mathcal{R}(A)$ is returned as the output of Algorithm 4.

Algorithm 4 `Right` algorithm of a non-terminal A

```
function RIGHT(A)
    R(A) ← ∅
    for each production of A do
        if A ⟶ γBβ and β ⟹* ε then
            R(A) ⊇ Right(B)
        end if
        if A ⟶ γaβ then
            a ∈ R(A)
        end if
    end for
    return R(A)
end function
```

Algorithm 5 receives a grammar $G(S) =< T, N, S, P >$ and Algorithms 2–4. Initially, precedence relations are added between the delimiting symbol \$ and the `Left` and `Right` sets of the non-terminal symbol of start S, according to rules `(A.1)` and `(A.2)`, respectively.

Algorithm 5 operates in an iterative manner. It traverses all the productions of the set P, and for each iteration, it initializes two temporary variables, u and l, to null and an empty list, respectively. It then performs a series of processes for each pair of contiguous grammatical symbols X and Y in a production.

1. The algorithm checks if XY has the form ab; if so, it adds the corresponding precedence relationship between a and b, following rule `(B)`.
2. The algorithm checks if XY has the form aB; if so, it assigns u the value of a and adds the corresponding precedence relations between a and terminals in `Left`(B), following rule `(C)`.
3. The algorithm checks if XY has the form Ab; if so, it adds the corresponding precedence relations between b and terminals in `Right`(A), following rule `(D.1)`. Then, check if $u \neq null$, and if so, add the corresponding precedence relationship between u and b, following rule `(D.2)`, and reset u to $null$. Finally, checks if list l has elements, and if so, adds the corresponding precedence relations between b and terminals in `Right`(B), following rule `(D.3)`, and resets list l to empty.

4. The algorithm checks if XY has the form AB; if so, it adds the corresponding precedence relations between terminals in $\texttt{Right}(A)$ and terminals in $\texttt{Leftmost}(B)$, following rule `(E.1)`. Then, it checks if $u \neq null$, and if so, adds the corresponding precedence relation between u and terminals in $\texttt{Left}(B)$, following rule `(E.2)`. Finally, it checks if the list l has elements and if so, adds the corresponding precedence relations between terminals in $\texttt{Right}(C)$, with $C \in l$ and terminals in $\texttt{Leftmost}(B)$, following rule `(E.3)`, and resets the list l if necessary.

Algorithm 5 Algorithm to obtain the operator precedence table of a G grammar

Require: $G(S) =< T, N, S, P >$
procedure TABLE(G, `Left`$(\cdot)$, `Right`$(\cdot)$, `Leftmost`$(\cdot)$)
compute `Left`(A), `Right`(A), `Leftmost`$(A), \forall A \in N$
add $\$ \lessdot a, \forall a \in \texttt{Left}(S)$ `(A.1)`
add $a \gtrdot \$, \forall a \in \texttt{Right}(S)$ `(A.2)`
for each $A \longrightarrow \gamma$ in P **do**
$u \leftarrow null$
$l \leftarrow [\,]$ // empty list
for each XY contiguous in γ **do**
if $XY = ab$ **then**
add $a \doteq b$ `(B)`
end if
if $XY = aB$ **then**
$u \leftarrow a$
add $a \lessdot b, \forall b \in \texttt{Left}(B)$ `(C)`
end if
if $XY = Ab$ **then**
add $a \gtrdot b, \forall a \in \texttt{Right}(A)$ `(D.1)`
if $u \neq null$ **then**
add $u \doteq b$ `(D.2)`
$u \leftarrow null$
end if
if $l \neq [\,]$ **then**
for $B \in l$ **do**
add $a \gtrdot b, \forall a \in \texttt{Right}(B)$ `(D.3)`
end for
$l \leftarrow [\,]$
end if
end if
if $XY = AB$ **then**
add $a \gtrdot b, \forall a \in \texttt{Right}(A) \wedge \forall b \in \texttt{Leftmost}(B)$ `(E.1)`
if $u \neq null$ **then**
add $u \lessdot b, \forall b \in \texttt{Left}(B)$ `(E.2)`
end if
if $l \neq [\,]$ **then**
for $C \in l$ **do**
add $c \gtrdot b, \forall c \in \texttt{Right}(C) \wedge \forall b \in \texttt{Leftmost}(B)$ `(E.3)`
end for
end if
if $B \stackrel{*}{\Longrightarrow} \varepsilon$ **then**
add A to l
else
$l \leftarrow [\,]$
end if
end if
end for
end for
end procedure

4.4. Complexity Analysis

The following variables will be established for the computation of the complexity of the four proposed algorithms:

- t : Cardinality of the set of terminal symbols.
- p : Cardinality of the set of productions.
- g : Number of grammatical symbols in the productions.
- n : Cardinality of the set of non-terminal symbols.

4.4.1. Complexity of Algorithm 2 (`Left`)

The `Left` function checks in which productions the symbol A appears on the left side. In the first **if-block**, a recursive call of `Left` is observed, which implies that all non-terminal symbols of G could be traversed at the end of the execution in the search for $\texttt{Left}(A)$. Therefore, all worst-case productions would be traversed, yielding a computational time of $\mathcal{O}(p)$. The second **if-block** has a computational time of $\mathcal{O}(1)$, which does not contribute to the overall computational time $\mathcal{O}(p)$.

4.4.2. Complexity of Algorithm 3 (`Leftmost`)

The `Leftmost` function has a similar structure to the previous one; therefore, the computational execution time is $\mathcal{O}(p)$.

4.4.3. Complexity of Algorithm 4 (`Right`)

The `Right` function has a similar structure to `Left`; therefore, the computational execution time is $\mathcal{O}(p)$.

4.4.4. Complexity of Algorithm 5 (`Table`)

According to the first **for-cycle**, the number of times it would be executed will be p. Immediately following the previous one, the inner **for-cycle** is executed in the worst-case g times, assuming that the total number of grammatical symbols appear in each production. Therefore, the computational cost of the two cycles is $\mathcal{O}(p \times g)$.

The computational costs for the construction of $\texttt{Left}(A)$, $\texttt{Right}(A)$, and $\texttt{Leftmost}(A)$, $\forall A \in N$, are already computed. In computing the complexity of `Table`, the length of each generated set will be considered, which in the worst case is t.

The analysis of the four **if-blocks** inside the second **for-cycle** is as follows:

- In the first **if-block**, after checking, the executed instruction is carried out in constant time and is denoted $\mathcal{O}(1)$.
- In the second **if-block**, after checking, the first instruction is executed in a constant time $\mathcal{O}(1)$, and the second is executed in a computational time $\mathcal{O}(t)$; therefore, the computational time of the **if-block** is $\mathcal{O}(t)$.
- In the third **if-block**, after checking, three sub-blocks are distinguished; the first one corresponds to the travel of $\texttt{Right}(A)$, executed in a computational time $\mathcal{O}(t)$. After checking $u \neq null$, the second sub-block presents two instructions executed in a constant time $\mathcal{O}(1)$. The third sub-block, after the checking of $l \neq [\,]$, presents a computational cost $\mathcal{O}(n \times t)$, determined by the for-cycle executed, in the worst case, $\mathcal{O}(n)$ times, corresponding to the number of non-terminal symbols in G and the travel it makes from $\texttt{Right}(B)$ with a cost $\mathcal{O}(t)$. In summary, the computational cost of the third **if-block** is $\mathcal{O}(n \times t)$, corresponding to the worst case.
- In the fourth **if-block**, after checking, four sub-blocks are distinguished. The first corresponds to the combinations formed from traversing $\texttt{Right}(A)$ and $\texttt{Leftmost}(B)$ to obtain the precedence relations $a \gtrdot b$. Since the maximum size of each set is t, the computational cost of these combinations is $\mathcal{O}(t^2)$. After checking $u \neq null$, the second sub-block presents an instruction where $\texttt{Left}(B)$ is traversed; therefore, the computational cost is $\mathcal{O}(t)$. In the third sub-block, after the checking of $l \neq [\,]$, it presents a **for-cycle** that executes at most n times the combinations formed from

traversing $\texttt{Right}(C)$ and $\texttt{Leftmost}(B)$ to obtain the precedence relations $c \gtrdot b$. These combinations are traversed maximally in $\mathcal{O}(t^2)$ computational time because t is the size of both sets and, consequently, the computational cost of the third sub-block is $\mathcal{O}(n \times t^2)$. The last sub-block will have a cost of $\mathcal{O}(1)$. In summary, the computational cost of the fourth block is $\mathcal{O}(n \times t^2)$, corresponding to the worst case.

In summary, the computational cost of the four **if-blocks** within the second **for-cycle** of Algorithm 5 is $\mathcal{O}(n \times t^2)$, this being the worst case; therefore, the computational cost of the main body of Algorithm 5 formed by the two nested **for-cycles** and the four **if-blocks** is $\mathcal{O}(p \times g \times n \times t^2)$ which is greater than the cost of computing any of the three sets of terminal symbols $\texttt{Left}(A)$, $\texttt{Right}(A)$, and $\texttt{Leftmost}(A)$, $\forall A \in N$, when starting the algorithm, and corresponding to $\mathcal{O}(p)$.

5. Examples

5.1. Example 1

5.1.1. Grammar

Given the following OPG:
$E \longrightarrow E + T \mid T$
$T \longrightarrow T * F \mid F$
$F \longrightarrow (E) \mid id$
The sets are built step by step following the algorithms.

5.1.2. Left

The $\texttt{Left}$ sets are calculated as detailed in Table 3.

Table 3. $\texttt{Left}$ Sets calculation for the CFG in Example 1.

$\texttt{Left}(E)$			
For $E \longrightarrow E + T$			
1. Evaluating the **first** conditional of Algorithm 2.	$A = E$ $\beta = \varepsilon$ $B = E$ $\gamma = +T$ $\mathcal{L}(E) \supseteq \mathcal{L}(E)$	2. Evaluating the **second** conditional of Algorithm 2.	$A = E$ $\beta = E$ $a = +$ $\gamma = T$ $\mathcal{L}(E) \supseteq \{+\}$
For $E \longrightarrow T$			
3. Evaluating the **first** conditional of Algorithm 2.	$A = E$ $\beta = \varepsilon$ $B = T$ $\gamma = \varepsilon$ $\mathcal{L}(E) = \{+\} \cup \mathcal{L}(T)$		

$\texttt{Left}(T)$			
For $T \longrightarrow T * F$			
1. Evaluating the **first** conditional of Algorithm 2.	$A = T$ $\beta = \varepsilon$ $B = T$ $\gamma = *F$ $\mathcal{L}(T) \supseteq \mathcal{L}(T)$	2. Evaluating the **second** conditional of Algorithm 2.	$A = T$ $\beta = T$ $a = *$ $\gamma = F$ $\mathcal{L}(T) \supseteq \{*\}$
For $T \longrightarrow F$			
3. Evaluating the **first** conditional of Algorithm 2.	$A = T$ $\beta = \varepsilon$ $B = F$ $\gamma = \varepsilon$ $\mathcal{L}(T) = \{*\} \cup \mathcal{L}(F)$		

Table 3. *Cont.*

`Left`(F)			
For $F \longrightarrow (E)$		For $F \longrightarrow id$	
1. Evaluating the **second** conditional of Algorithm 2.	$A = F$ $\beta = \varepsilon$ $a = ($ $\gamma = E)$ $\mathcal{L}(F) \supseteq \{(\}$	2. Evaluating the **second** conditional of Algorithm 2.	$A = F$ $\beta = \varepsilon$ $a = id$ $\gamma = \varepsilon$ $\mathcal{L}(F) = \{(, id\}$

Summary of `Left` sets of non-terminals:

- $\texttt{Left}(E) = \{+\} \cup \mathcal{L}(T)$.
- $\texttt{Left}(T) = \{*\} \cup \mathcal{L}(F)$.
- $\texttt{Left}(F) = \{(, id\}$.

Substituting the sets:

- $\texttt{Left}(E) = \{+, *, (, id\}$.
- $\texttt{Left}(T) = \{*, (, id\}$.
- $\texttt{Left}(F) = \{(, id\}$.

5.1.3. Right

The `Right` sets are calculated as detailed in Table 4.

Table 4. `Right` Sets calculation for the CFG in Example 1.

`Right`(E)			
For $E \longrightarrow E + T$			
1. Evaluating the **first** conditional of Algorithm 4.	$A = E$ $\gamma = E+$ $B = T$ $\beta = \varepsilon$ $\mathcal{R}(E) \supseteq \mathcal{R}(T)$	2. Evaluating the **second** conditional of Algorithm 4.	$A = E$ $\gamma = E$ $a = +$ $\beta = T$ $\mathcal{R}(E) \supseteq \mathcal{R}(T) \cup \{+\}$
For $E \longrightarrow T$			
3. Evaluating the **first** conditional of Algorithm 4.	$A = E$ $\gamma = \varepsilon$ $B = T$ $\beta = \varepsilon$ $\mathcal{R}(E) = \mathcal{R}(T) \cup \{+\}$		

`Right`(T)			
For $T \longrightarrow T * F$			
1. Evaluating the **first** conditional of Algorithm 4.	$A = T$ $\gamma = T*$ $B = F$ $\beta = \varepsilon$ $\mathcal{R}(T) \supseteq \mathcal{R}(F)$	2. Evaluating the **second** conditional of Algorithm 4.	$A = T$ $\gamma = T$ $a = *$ $\beta = F$ $\mathcal{R}(T) \supseteq \mathcal{R}(F) \cup \{*\}$
For $T \longrightarrow F$			
3. Evaluating the **first** conditional of Algorithm 4.	$A = T$ $\gamma = \varepsilon$ $B = F$ $\beta = \varepsilon$ $\mathcal{R}(T) = \mathcal{R}(F) \cup \{*\}$		

Table 4. *Cont.*

`Right`(F)			
For $F \longrightarrow (E)$		For $F \longrightarrow id$	
1. Evaluating the **second** conditional of Algorithm 2.	$A = F$ $\gamma = (E$ $a =)$ $\beta = \varepsilon$ $\mathcal{R}(F) \supseteq \{)\}$	2. Evaluating the **second** conditional of Algorithm 2.	$A = F$ $\gamma = \varepsilon$ $a = id$ $\beta = \varepsilon$ $\mathcal{R}(F) = \{), id\}$

Summary of `Right` sets of non-terminals:

- $\texttt{Right}(E) = \mathcal{R}(T) \cup \{+\}$.
- $\texttt{Right}(T) = \mathcal{R}(F) \cup \{*\}$.
- $\texttt{Right}(F) = \{), id\}$.

Substituting the sets:

- $\texttt{Right}(E) = \{), id, *, +\}$.
- $\texttt{Right}(T) = \{), id, *\}$.
- $\texttt{Right}(F) = \{), id\}$.

5.1.4. Leftmost

The `Leftmost` are calculated as detailed in Table 5.

Table 5. `Leftmost` Sets calculation for the CFG in Example 1.

`Leftmost`(E)			
For $E \longrightarrow E + T$		For $E \longrightarrow T$	
1. Evaluating the **first** conditional of Algorithm 3.	$A = E$ $\beta = \varepsilon$ $B = E$ $\gamma = +T$ $\mathcal{L}_m(E) \supseteq \mathcal{L}_m(E)$	2. Evaluating the **first** conditional of Algorithm 3.	$A = E$ $\beta = \varepsilon$ $B = T$ $\gamma = \varepsilon$ $\mathcal{L}_m(E) = \mathcal{L}_m(T)$

`Leftmost`(T)			
For $T \longrightarrow T * F$		For $T \longrightarrow F$	
1. Evaluating the **first** conditional of Algorithm 3.	$A = T$ $\beta = \varepsilon$ $B = T$ $\gamma = *F$ $\mathcal{L}_m(T) \supseteq \mathcal{L}_m(T)$	2. Evaluating the **first** conditional of Algorithm 3.	$A = T$ $\beta = \varepsilon$ $B = F$ $\gamma = \varepsilon$ $\mathcal{L}_m(T) = \mathcal{L}_m(F)$

`Leftmost`(F)			
For $F \longrightarrow (E)$		For $F \longrightarrow id$	
1. Evaluating the **second** conditional of Algorithm 3.	$A = F$ $\beta = \varepsilon$ $a = ($ $\gamma = E)$ $\mathcal{L}_m(F) \supseteq \{(\}$	2. Evaluating the **second** conditional of Algorithm 3.	$A = F$ $\beta = \varepsilon$ $a = id$ $\gamma = \varepsilon$ $\mathcal{L}_m(F) = \{(, id\}$

Summary of `Leftmost` sets of non-terminals:

- $\texttt{Leftmost}(E) = \mathcal{L}_m(T)$.

- $\texttt{Leftmost}(T) = \mathcal{L}_m(F)$.
- $\texttt{Leftmost}(F) = \{(, id\}$.

Substituting the sets:

- $\texttt{Leftmost}(E) = \{(, id\}$.
- $\texttt{Leftmost}(T) = \{(, id\}$.
- $\texttt{Leftmost}(F) = \{(, id\}$.

5.1.5. Case Studies

For Algorithm 5, the application of some precedence relations is exhibited.

1. For the delimiter symbol \$:
   (a) $\$ \lessdot \mathcal{L}(E) \rightarrow$ Rule `(A.1)`.
   - $\$ \lessdot +$
   - $\$ \lessdot *$
   - $\$ \lessdot ($
   - $\$ \lessdot id$

   (b) $\mathcal{R}(E) \gtrdot \$ \rightarrow$ Rule `(A.2)`.
   - $) \gtrdot \$$
   - $id \gtrdot \$$
   - $* \gtrdot \$$
   - $+ \gtrdot \$$
2. For $E \longrightarrow E + T$
   (a) $X = E$, $Y = +$ $\rightarrow$ Rule `(D.1)`.
   $\mathcal{R}(E) \gtrdot +$
   - $) \gtrdot +$
   - $id \gtrdot +$
   - $* \gtrdot +$
   - $+ \gtrdot +$

   $u = null$, $l = [\,]$
   (b) $X = +$, $Y = T$ $\rightarrow$ Rule `(C)`.
   $+ \lessdot \mathcal{L}(T)$
   - $+ \lessdot *$
   - $+ \lessdot ($
   - $+ \lessdot id$

   $u = +$, $l = [\,]$
3. For $F \longrightarrow (E)$
   (a) $X = ($, $Y = E$ $\rightarrow$ Rule `(C)`.
   $(\lessdot \mathcal{L}(E)$
   - $(\lessdot +$
   - $(\lessdot *$
   - $(\lessdot ($
   - $(\lessdot id$

   $u = ($, $l = [\,]$
   (b) $X = E$, $Y =)$ $\rightarrow$ Rule `(D.1)`.
   $\mathcal{R}(E) \gtrdot)$
   - $) \gtrdot)$
   - $id \gtrdot)$
   - $* \gtrdot)$
   - $+ \gtrdot)$

   $u \neq null \rightarrow u \doteq b \rightarrow (\doteq)$ $\rightarrow$ Rule `(D.2)`.
   $u = null$, $l = [\,]$

4. In the following cases, the form XY is not fulfilled in the right side part of the production; therefore, no rule can be applied:
   (a) $E \longrightarrow T$
   (b) $T \longrightarrow F$
   (c) $F \longrightarrow id$

The remaining precedence relations are derived using the same method as the previously demonstrated cases.

5.1.6. Operator Precedence Table

At the end of the algorithm execution, operator precedence Table 6 is obtained.

Table 6. Precedence Table of Example 1.

	$+$	$*$	$($	$)$	id	$\$$
$+$	$\gtrdot$	$\lessdot$	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
$*$	$\gtrdot$	$\gtrdot$	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
$($	$\lessdot$	$\lessdot$	$\lessdot$	$\doteq$	$\lessdot$	
$)$	$\gtrdot$	$\gtrdot$		$\gtrdot$		$\gtrdot$
id	$\gtrdot$	$\gtrdot$		$\gtrdot$		$\gtrdot$
$\$$	$\lessdot$	$\lessdot$	$\lessdot$		$\lessdot$	

5.1.7. Bottom-Up Parsing

Let $id + ((id + id) * (id)) * id$ be a string obtained by rightmost derivation, then applying Algorithm 1 results in the bottom-up parsing detailed in Table 7.

Table 7. Bottom-up Parsing of Example 1.

Stack	Input	Action
$\$$	$id + ((id + id) * (id)) * id\$$	Shift
$\$id$	$+((id + id) * (id)) * id\$$	Reduce
$\$$	$+((id + id) * (id)) * id\$$	Shift
$\$+$	$((id + id) * (id)) * id\$$	Shift
$\$ + ($	$(id + id) * (id)) * id\$$	Shift
$\$ + (($	$id + id) * (id)) * id\$$	Shift
$\$ + ((id$	$+id) * (id)) * id\$$	Reduce
$\$ + (($	$+id) * (id)) * id\$$	Shift
$\$ + ((+$	$id) * (id)) * id\$$	Shift
$\$ + ((+id$	$) * (id)) * id\$$	Reduce
$\$ + ((+$	$) * (id)) * id\$$	Reduce
$\$ + (($	$) * (id)) * id\$$	Shift
$\$ + (()$	$*(id)) * id\$$	Reduce
$\$ + ($	$*(id)) * id\$$	Shift
$\$ + (*$	$(id)) * id\$$	Shift
$\$ + (*($	$id)) * id\$$	Shift
$\$ + (*(id$	$)) * id\$$	Reduce
$\$ + (*($	$)) * id\$$	Shift

Table 7. *Cont.*

Stack	Input	Action
$\$ + (*()$	$) * id\$$	Reduce
$\$ + (*$	$) * id\$$	Reduce
$\$ + ($	$) * id\$$	Shift
$\$ + ()$	$*id\$$	Reduce
$\$+$	$*id\$$	Shift
$\$ + *$	$id\$$	Shift
$\$ + *id$	$\$$	Reduce
$\$ + *$	$\$$	Reduce
$\$+$	$\$$	Reduce
$\$$	$\$$	***Accept***

5.2. Example 2

5.2.1. Grammar

Given the following non-OPG:
$S \longrightarrow SD; \mid D;$
$D \longrightarrow Tid(L)$
$T \longrightarrow T* \mid int$
$L \longrightarrow I \mid \varepsilon$
$I \longrightarrow T \mid T, I$
The sets are built step by step following the algorithms.

5.2.2. Left

The `Left` sets are calculated as detailed in Table 8.

Table 8. `Left` Sets calculation for the CFG in Example 2.

`Left(`S`)`			
For $S \longrightarrow SD;$			
1. Evaluating the **first** conditional of Algorithm 2.	$A = S$ $\beta = \varepsilon$ $B = S$ $\gamma = D;$ $\mathcal{L}(S) \supseteq \mathcal{L}(S)$	2. Evaluating the **first** conditional of Algorithm 2.	$A = S$ $\beta = S$ $B = D$ $\gamma =;$ $\mathcal{L}(S) \supseteq \mathcal{L}(D)$
3. Evaluating the **second** conditional of Algorithm 2.	$A = S$ $\beta = SD$ $a =;$ $\gamma = \varepsilon$ $\mathcal{L}(S) \supseteq \mathcal{L}(D) \cup \{;\}$		
For $S \longrightarrow D;$			
4. Evaluating the **first** conditional of Algorithm 2.	$A = S$ $\beta = \varepsilon$ $B = D$ $\gamma =;$ $\mathcal{L}(S) \supseteq \mathcal{L}(D) \cup \{;\} \cup \mathcal{L}(D)$	5. Evaluating the **second** conditional of Algorithm 2.	$A = S$ $\beta = D$ $a =;$ $\gamma = \varepsilon$ $\mathcal{L}(S) = \mathcal{L}(D) \cup \{;\}$

Table 8. *Cont.*

`Left`(D)			
For $D \longrightarrow Tid(L)$			
1. Evaluating the **first** conditional of Algorithm 2.	$A = D$ $\beta = \varepsilon$ $B = T$ $\gamma = id(L)$ $\mathcal{L}(D) \supseteq \mathcal{L}(T)$	2. Evaluating the **second** conditional of Algorithm 2.	$A = D$ $\beta = T$ $a = id$ $\gamma = (L)$ $\mathcal{L}(D) = \mathcal{L}(T) \cup \{id\}$

`Left`(T)			
For $T \longrightarrow T*$			
1. Evaluating the **first** conditional of Algorithm 2.	$A = T$ $\beta = \varepsilon$ $B = T$ $\gamma = *$ $\mathcal{L}(T) \supseteq \mathcal{L}(T)$	2. Evaluating the **second** conditional of Algorithm 2.	$A = T$ $\beta = T$ $a = *$ $\gamma = \varepsilon$ $\mathcal{L}(T) \supseteq \{*\}$
For $T \longrightarrow int$			
3. Evaluating the **second** conditional of Algorithm 2.	$A = T$ $\beta = \varepsilon$ $a = int$ $\gamma = \varepsilon$ $\mathcal{L}(T) = \{*, int\}$		

`Left`(L)			
For $L \longrightarrow I$			
1. Evaluating the **first** conditional of Algorithm 2.	$A = L$ $\beta = \varepsilon$ $B = I$ $\gamma = \varepsilon$ $\mathcal{L}(L) = \mathcal{L}(I)$		

`Left`(I)			
For $I \longrightarrow T$			
1. Evaluating the **first** conditional of Algorithm 2.	$A = I$ $\beta = \varepsilon$ $B = T$ $\gamma = \varepsilon$ $\mathcal{L}(I) \supseteq \mathcal{L}(T)$		
For $I \longrightarrow T, I$			
2. Evaluating the **first** conditional of Algorithm 2.	$A = I$ $\beta = \varepsilon$ $B = T$ $\gamma =, I$ $\mathcal{L}(I) \supseteq \mathcal{L}(T)$	3. Evaluating the **second** conditional of Algorithm 2.	$A = I$ $\beta = T$ $a =,$ $\gamma = I$ $\mathcal{L}(I) = \mathcal{L}(T) \cup \{,\}$

Summary of `Left` sets of non-terminals:

- `Left`$(S) = \mathcal{L}(D) \cup \{;\}$.
- `Left`$(D) = \mathcal{L}(T) \cup \{id\}$.
- `Left`$(T) = \{*, int\}$.
- `Left`$(L) = \mathcal{L}(I)$.
- `Left`$(I) = \mathcal{L}(T) \cup \{,\}$.

Substituting the sets:

- $\texttt{Left}(S) = \{;, id, *, int\}$.
- $\texttt{Left}(D) = \{id, *, int\}$.
- $\texttt{Left}(T) = \{*, int\}$.
- $\texttt{Left}(L) = \{,, *, int\}$.
- $\texttt{Left}(I) = \{,, *, int\}$.

5.2.3. Right

The `Right` sets are calculated as detailed in Table 9.

Table 9. `Right` Sets calculation for the CFG in Example 2.

`Right`(S)			
For $S \longrightarrow SD$;		For $S \longrightarrow D$;	
1. Evaluating the **second** conditional of Algorithm 4.	$A = S$ $\gamma = SD$ $a = ;$ $\beta = \varepsilon$ $\mathcal{R}(S) \supseteq \{;\}$	2. Evaluating the **second** conditional of Algorithm 4.	$A = S$ $\gamma = D$ $a = ;$ $\beta = \varepsilon$ $\mathcal{R}(S) = \{;\}$

`Right`(D)	
For $D \longrightarrow Tid(L)$	
1. Evaluating the **second** conditional of Algorithm 4.	$A = D$ $\gamma = Tid(L$ $a =)$ $\beta = \varepsilon$ $\mathcal{R}(D) = \{)\}$

`Right`(T)			
For $T \longrightarrow T*$		For $T \longrightarrow int$	
1. Evaluating the **second** conditional of Algorithm 4.	$A = T$ $\gamma = T$ $a = *$ $\beta = \varepsilon$ $\mathcal{R}(T) \supseteq \{*\}$	2. Evaluating the **second** conditional of Algorithm 4.	$A = T$ $\gamma = \varepsilon$ $a = int$ $\beta = \varepsilon$ $\mathcal{R}(T) = \{*, int\}$

`Right`(L)	
For $L \longrightarrow I$	
1. Evaluating the **first** conditional of Algorithm 4.	$A = L$ $\gamma = \varepsilon$ $B = I$ $\beta = \varepsilon$ $\mathcal{R}(L) = \mathcal{R}(I)$

Table 9. *Cont.*

`Right`(I)			
For $I \longrightarrow T$			
1. Evaluating the **first** conditional of Algorithm 4.	$A = I$ $\gamma = \varepsilon$ $B = T$ $\beta = \varepsilon$ $\mathcal{R}(I) \supseteq \mathcal{R}(T)$		
For $I \longrightarrow T, I$			
2. Evaluating the **first** conditional of Algorithm 4.	$A = I$ $\gamma = T,$ $B = I$ $\beta = \varepsilon$ $\mathcal{R}(I) \supseteq \mathcal{R}(T) \cup \mathcal{R}(I)$	3. Evaluating the **second** conditional of Algorithm 4.	$A = I$ $\gamma = T$ $a =,$ $\beta = I$ $\mathcal{R}(I) = \mathcal{R}(T) \cup \{,\}$

Summary of `Right` sets of non-terminals:

- `Right`$(S) = \{;\}$.
- `Right`$(D) = \{)\}$.
- `Right`$(T) = \{*, int\}$.
- `Right`$(L) = \mathcal{R}(I)$.
- `Right`$(I) = \mathcal{R}(T) \cup \{,\}$.

Substituting the sets:

- `Right`$(S) = \{;\}$.
- `Right`$(D) = \{)\}$.
- `Right`$(T) = \{*, int\}$.
- `Right`$(L) = \{,, *, int\}$.
- `Right`$(I) = \{,, *, int\}$.

5.2.4. Leftmost

The `Leftmost` sets are calculated as detailed in Table 10.

Table 10. `Leftmost` Sets calculation for the CFG in Example 2.

`Leftmost`(S)			
For $S \longrightarrow SD;$		For $S \longrightarrow D;$	
1. Evaluating the **first** conditional of Algorithm 3.	$A = S$ $\beta = \varepsilon$ $B = S$ $\gamma = D;$ $\mathcal{L}_m(S) \supseteq \mathcal{L}_m(S)$	2. Evaluating the **first** conditional of Algorithm 3.	$A = S$ $\beta = \varepsilon$ $B = D$ $\gamma =;$ $\mathcal{L}_m(S) = \mathcal{L}_m(D)$

`Leftmost`(D)			
For $D \longrightarrow Tid(L)$			
1. Evaluating the **first** conditional of Algorithm 3.	$A = D$ $\beta = \varepsilon$ $B = T$ $\gamma = id(L)$ $\mathcal{L}_m(D) = \mathcal{L}_m(T)$		

Table 10. *Cont.*

`Leftmost`(T)			
For $T \longrightarrow T*$		For $T \longrightarrow int$	
1. Evaluating the **first** conditional of Algorithm 3.	$A = T$ $\beta = \varepsilon$ $B = T$ $\gamma = *$ $\mathcal{L}_m(T) \supseteq \mathcal{L}_m(T)$	2. Evaluating the **second** conditional of Algorithm 3.	$A = T$ $\beta = \varepsilon$ $a = int$ $\gamma = \varepsilon$ $\mathcal{L}_m(T) = \{int\}$

`Leftmost`(L)	
For $L \longrightarrow I$	
1. Evaluating the **first** conditional of Algorithm 3.	$A = L$ $\beta = \varepsilon$ $B = I$ $\gamma = \varepsilon$ $\mathcal{L}_m(L) = \mathcal{L}_m(I)$

`Leftmost`(I)			
For $I \longrightarrow T$		For $I \longrightarrow T, I$	
1. Evaluating the **first** conditional of Algorithm 3.	$A = I$ $\beta = \varepsilon$ $B = T$ $\gamma = \varepsilon$ $\mathcal{L}_m(I) \supseteq \mathcal{L}_m(T)$	2. Evaluating the **first** conditional of Algorithm 3.	$A = I$ $\beta = \varepsilon$ $B = T$ $\gamma =, I$ $\mathcal{L}_m(I) = \mathcal{L}_m(T)$

Summary of `Leftmost` sets of non-terminals:

- $\texttt{Leftmost}(S) = \mathcal{L}_m(D)$.
- $\texttt{Leftmost}(D) = \mathcal{L}_m(T)$.
- $\texttt{Leftmost}(T) = \{int\}$.
- $\texttt{Leftmost}(L) = \mathcal{L}_m(I)$.
- $\texttt{Leftmost}(I) = \mathcal{L}_m(T)$.

Substituting the sets:

- $\texttt{Leftmost}(S) = \{int\}$.
- $\texttt{Leftmost}(D) = \{int\}$.
- $\texttt{Leftmost}(T) = \{int\}$.
- $\texttt{Leftmost}(L) = \{int\}$.
- $\texttt{Leftmost}(I) = \{int\}$.

5.2.5. Case Studies

For Algorithm 5, the application of some precedence relations is exhibited.

1. For the delimiter symbol \$:
   (a) $\$ \lessdot \mathcal{L}(S) \rightarrow$ Rule `(A.1)`.
   - $\$ \lessdot\ ;$
   - $\$ \lessdot id$
   - $\$ \lessdot *$
   - $\$ \lessdot int$

   (b) $\mathcal{R}(S) \gtrdot \$ \rightarrow$ Rule `(A.2)`.
   - $;\ \gtrdot \$$

2. For $S \longrightarrow SD$;

(a) $X = S$, $Y = D$ $\rightarrow$ Rule (E.1).
$\mathcal{R}(S) \gtrdot \mathcal{L}_m(D)$
- $; \gtrdot int$

$u = null$, $l = [\,]$

(b) $X = D$, $Y = ;$ $\rightarrow$ Rule (D.1).
$\mathcal{R}(D) \gtrdot ;$
- $) \gtrdot ;$

$u = null$, $l = [\,]$

3. For $D \longrightarrow Tid(L)$

(a) $X = id$, $Y = ($ $\rightarrow$ Rule (B).
- $id \doteq ($

$u = null$, $l = [\,]$

(b) $X = ($, $Y = L$ $\rightarrow$ Rule (C).
$(\lessdot \mathcal{L}(L)$
- $(\lessdot ,$
- $(\lessdot *$
- $(\lessdot int$

$u = ($, $l = [\,]$

(c) $X = L$, $Y =)$ $\rightarrow$ Rule (D.1).
$\mathcal{R}(L) \gtrdot)$
- $, \gtrdot)$
- $* \gtrdot)$
- $int \gtrdot)$

$u \neq null \rightarrow u \doteq b \rightarrow (\doteq)$ $\rightarrow$ Rule (D.2).
$u = null$, $l = [\,]$

4. In the following cases, the form XY is not fulfilled in the right side part of the production; therefore, no rule can be applied:

(a) $T \longrightarrow int$
(b) $L \longrightarrow I$
(c) $L \longrightarrow \varepsilon$
(d) $I \longrightarrow T$

The remaining precedence relations are derived using the same method as the previously demonstrated cases.

5.2.6. Operator Precedence Table

At the end of the algorithm execution, operator precedence Table 11 is obtained.

Table 11. Precedence Table of Example 2.

	;	*id*	(	)	*	*int*	,	$
;						$\gtrdot$		$\gtrdot$
id			$\doteq$					
(				$\doteq$	$\lessdot$	$\lessdot$	$\lessdot$	
)	$\gtrdot$							
*		$\gtrdot$		$\gtrdot$	$\gtrdot$		$\gtrdot$	
int		$\gtrdot$		$\gtrdot$	$\gtrdot$		$\gtrdot$	
,				$\gtrdot$	$\lessdot$	$\lessdot$	$\lessdot$	
$	$\lessdot$	$\lessdot$			$\lessdot$	$\lessdot$		

5.2.7. Bottom-Up Parsing

Let $int\ id();int\ id(int,int);$ be a string obtained by rightmost derivation, then applying Algorithm 1 results in the bottom-up parsing detailed in Table 12.

Table 12. Bottom-up Parsing of Example 2.

Stack	Input	Action
\$	$int\ id();int\ id(int,int);\$$	Shift
\$$int$	$id();int\ id(int,int);\$$	Reduce
\$	$id();int\ id(int,int);\$$	Shift
\$$id$	$();int\ id(int,int);\$$	Shift
\$$id($	$);int\ id(int,int);\$$	Shift
\$$id()$	$;int\ id(int,int);\$$	Reduce
\$	$;int\ id(int,int);\$$	Shift
\$;	$int\ id(int,int);\$$	Reduce
\$	$int\ id(int,int);\$$	Shift
\$$int$	$id(int,int);\$$	Reduce
\$	$id(int,int);\$$	Shift
\$$id$	$(int,int);\$$	Shift
\$$id($	$int,int);\$$	Shift
\$$id(int$	$,int);\$$	Reduce
\$$id($	$,int);\$$	Shift
\$$id(,$	$int);\$$	Shift
\$$id(,int$	$);\$$	Reduce
\$$id(,$	$);\$$	Reduce
\$$id($	$);\$$	Shift
\$$id()$	$;\$$	Reduce
\$	$;\$$	Shift
\$;	\$	Reduce
\$	\$	***Accept***

5.3. Example 3

5.3.1. Grammar

Given the following non-OPG:
$S \longrightarrow ABC$
$A \longrightarrow aA \mid a$
$B \longrightarrow bB \mid b \mid \varepsilon$
$C \longrightarrow CDc \mid c$
$D \longrightarrow d$
The sets are built step by step following the algorithms.

5.3.2. Left

The `Left` sets are calculated as detailed in Table 13.

Table 13. `Left` Sets calculation for the CFG in Example 3.

`Left`(S)			
For $S \longrightarrow ABC$			
1. Evaluating the **first** conditional of Algorithm 2.	$A = S$ $\beta = \varepsilon$ $B = A$ $\gamma = BC$ $\mathcal{L}(S) \supseteq \mathcal{L}(A)$	2. Evaluating the **first** conditional of Algorithm 2.	$A = S$ $\beta = A$ $B = B$ $\gamma = C$ $\mathcal{L}(S) \supseteq \mathcal{L}(A) \cup \mathcal{L}(B)$
3. Evaluating the **first** conditional of Algorithm 2.	$A = S$ $\beta = AB$ $B = C$ $\gamma = \varepsilon$ $\mathcal{L}(S) = \mathcal{L}(A) \cup \mathcal{L}(B) \cup \mathcal{L}(C)$		

`Left`(A)			
For $A \longrightarrow aA$			
1. Evaluating the **second** conditional of Algorithm 2.	$A = A$ $\beta = \varepsilon$ $a = a$ $\gamma = A$ $\mathcal{L}(A) \supseteq \{a\}$	2. Evaluating the **first** conditional of Algorithm 2.	$A = A$ $\beta = a$ $B = A$ $\gamma = \varepsilon$ $\mathcal{L}(A) \supseteq \{a\} \cup \mathcal{L}(A)$
For $A \longrightarrow a$			
3. Evaluating the **second** conditional of Algorithm 2.	$A = A$ $\beta = \varepsilon$ $a = a$ $\gamma = \varepsilon$ $\mathcal{L}(A) = \{a\}$		

`Left`(B)			
For $B \longrightarrow bB$			
1. Evaluating the **second** conditional of Algorithm 2.	$A = B$ $\beta = \varepsilon$ $a = b$ $\gamma = B$ $\mathcal{L}(B) \supseteq \{b\}$	2. Evaluating the **first** conditional of Algorithm 2.	$A = B$ $\beta = b$ $B = B$ $\gamma = \varepsilon$ $\mathcal{L}(B) \supseteq \{b\} \cup \mathcal{L}(B)$
For $B \longrightarrow b$			
3. Evaluating the **second** conditional of Algorithm 2.	$A = B$ $\beta = \varepsilon$ $a = b$ $\gamma = \varepsilon$ $\mathcal{L}(B) = \{b\}$		

Table 13. *Cont.*

`Left`(C)			
For $C \longrightarrow CDc$			
1. Evaluating the **first** conditional of Algorithm 2.	$A = C$ $\beta = \varepsilon$ $B = C$ $\gamma = Dc$ $\mathcal{L}(C) \supseteq \mathcal{L}(C)$	2. Evaluating the **first** conditional of Algorithm 2.	$A = C$ $\beta = C$ $B = D$ $\gamma = c$ $\mathcal{L}(C) \supseteq \mathcal{L}(D)$
3. Evaluating the **second** conditional of Algorithm 2.	$A = C$ $\beta = CD$ $a = c$ $\gamma = \varepsilon$ $\mathcal{L}(C) \supseteq \mathcal{L}(D) \cup \{c\}$		
For $C \longrightarrow c$			
4. Evaluating the **second** conditional of Algorithm 2.	$A = C$ $\beta = \varepsilon$ $a = c$ $\gamma = \varepsilon$ $\mathcal{L}(C) = \mathcal{L}(D) \cup \{c\}$		

`Left`(D)	
For $D \longrightarrow d$	
1. Evaluating the **second** conditional of Algorithm 2.	$A = D$ $\beta = \varepsilon$ $a = d$ $\gamma = \varepsilon$ $\mathcal{L}(D) = \{d\}$

Summary of `Left` sets of non-terminals:

- $\texttt{Left}(S) = \mathcal{L}(A) \cup \mathcal{L}(B) \cup \mathcal{L}(C)$.
- $\texttt{Left}(A) = \{a\}$.
- $\texttt{Left}(B) = \{b\}$.
- $\texttt{Left}(C) = \mathcal{L}(D) \cup \{c\}$.
- $\texttt{Left}(D) = \{d\}$.

Substituting the sets:

- $\texttt{Left}(S) = \{a, b, d, c\}$.
- $\texttt{Left}(A) = \{a\}$.
- $\texttt{Left}(B) = \{b\}$.
- $\texttt{Left}(C) = \{d, c\}$.
- $\texttt{Left}(D) = \{d\}$.

5.3.3. Right

The `Right` sets are calculated as detailed in Table 14

Table 14. `Right` Sets calculation for the CFG in Example 3.

`Right`(S)			
For $S \longrightarrow ABC$			
1. Evaluating the **first** conditional of Algorithm 4.	$A = S$ $\gamma = AB$ $B = C$ $\beta = \varepsilon$ $\mathcal{R}(S) = \mathcal{R}(C)$		

`Right`(A)			
For $A \longrightarrow aA$			
1. Evaluating the **first** conditional of Algorithm 4.	$A = A$ $\gamma = a$ $B = A$ $\beta = \varepsilon$ $\mathcal{R}(A) \supseteq \mathcal{R}(A)$	2. Evaluating the **second** conditional of Algorithm 4.	$A = A$ $\gamma = \varepsilon$ $a = a$ $\beta = A$ $\mathcal{R}(A) \supseteq \{a\}$
For $A \longrightarrow a$			
3. Evaluating the **second** conditional of Algorithm 4.	$A = A$ $\gamma = \varepsilon$ $a = a$ $\beta = \varepsilon$ $\mathcal{R}(A) = \{a\}$		

`Right`(B)			
For $B \longrightarrow bB$			
1. Evaluating the **first** conditional of Algorithm 4.	$A = B$ $\gamma = b$ $B = B$ $\beta = \varepsilon$ $\mathcal{R}(B) \supseteq \mathcal{R}(B)$	2. Evaluating the **second** conditional of Algorithm 4.	$A = B$ $\gamma = \varepsilon$ $a = b$ $\beta = B$ $\mathcal{R}(B) \supseteq \{b\}$
For $B \longrightarrow b$			
3. Evaluating the **second** conditional of Algorithm 4.	$A = B$ $\gamma = \varepsilon$ $a = b$ $\beta = \varepsilon$ $\mathcal{R}(b) = \{b\}$		

`Right`(C)			
For $C \longrightarrow CDc$		For $C \longrightarrow c$	
1. Evaluating the **second** conditional of Algorithm 4.	$A = C$ $\gamma = CD$ $a = c$ $\beta = \varepsilon$ $\mathcal{R}(C) \supseteq \{c\}$	2. Evaluating the **second** conditional of Algorithm 4.	$A = C$ $\gamma = \varepsilon$ $a = c$ $\beta = \varepsilon$ $\mathcal{R}(C) = \{c\}$

Table 14. *Cont.*

`Right`(D)	
For $D \longrightarrow d$	
1. Evaluating the **second** conditional of Algorithm 4.	$A = D$ $\gamma = \varepsilon$ $a = d$ $\beta = \varepsilon$ $\mathcal{R}(D) = \{d\}$

Summary of `Right` sets of non-terminals:

- $\texttt{Right}(S) = \mathcal{R}(C)$.
- $\texttt{Right}(A) = \{a\}$.
- $\texttt{Right}(B) = \{b\}$.
- $\texttt{Right}(C) = \{c\}$.
- $\texttt{Right}(D) = \{d\}$.

Substituting the sets:

- $\texttt{Right}(S) = \{c\}$.
- $\texttt{Right}(A) = \{a\}$.
- $\texttt{Right}(B) = \{b\}$.
- $\texttt{Right}(C) = \{c\}$.
- $\texttt{Right}(D) = \{d\}$.

5.3.4. Leftmost

The `Leftmost` sets are calculated as detailed in Table 15.

Table 15. `Leftmost` Sets calculation for the CFG in Example 3.

`Leftmost`(S)	
For $S \longrightarrow ABC$	
1. Evaluating the **first** conditional of Algorithm 3.	$A = S$ $\beta = \varepsilon$ $B = A$ $\gamma = BC$ $\mathcal{L}_m(S) = \mathcal{L}_m(A)$

`Leftmost`(A)			
For $A \longrightarrow aA$		For $A \longrightarrow a$	
1. Evaluating the **second** conditional of Algorithm 3.	$A = A$ $\beta = \varepsilon$ $a = a$ $\gamma = A$ $\mathcal{L}_m(A) \supseteq \{a\}$	2. Evaluating the **second** conditional of Algorithm 3.	$A = A$ $\beta = \varepsilon$ $a = a$ $\gamma = \varepsilon$ $\mathcal{L}_m(A) = \{a\}$

`Leftmost`(B)			
For $B \longrightarrow bB$		For $B \longrightarrow b$	
1. Evaluating the **second** conditional of Algorithm 3.	$A = B$ $\beta = \varepsilon$ $a = b$ $\gamma = B$ $\mathcal{L}_m(B) \supseteq \{b\}$	2. Evaluating the **second** conditional of Algorithm 3.	$A = B$ $\beta = \varepsilon$ $a = b$ $\gamma = \varepsilon$ $\mathcal{L}_m(B) = \{b\}$

Table 15. *Cont.*

`Leftmost`(*C*)			
For $C \longrightarrow CDc$		For $C \longrightarrow c$	
1. Evaluating the **first** conditional of Algorithm 3.	$A = C$ $\beta = \varepsilon$ $B = C$ $\gamma = Dc$ $\mathcal{L}_m(C) \supseteq \mathcal{L}_m(C)$	2. Evaluating the **second** conditional of Algorithm 3.	$A = C$ $\beta = \varepsilon$ $a = c$ $\gamma = \varepsilon$ $\mathcal{L}_m(C) = \{c\}$

`Leftmost`(*D*)	
For $D \longrightarrow d$	
1. Evaluating the **second** conditional of Algorithm 3.	$A = D$ $\beta = \varepsilon$ $a = d$ $\gamma = \varepsilon$ $\mathcal{L}_m(D) = \{d\}$

Summary of `Leftmost` sets of non-terminals:

- $\texttt{Leftmost}(S) = \mathcal{L}_m(A)$.
- $\texttt{Leftmost}(A) = \{a\}$.
- $\texttt{Leftmost}(B) = \{b\}$.
- $\texttt{Leftmost}(C) = \{c\}$.
- $\texttt{Leftmost}(D) = \{d\}$.

Substituting the sets:

- $\texttt{Leftmost}(S) = \{a\}$.
- $\texttt{Leftmost}(A) = \{a\}$.
- $\texttt{Leftmost}(B) = \{b\}$.
- $\texttt{Leftmost}(C) = \{c\}$.
- $\texttt{Leftmost}(D) = \{d\}$.

5.3.5. Case Studies

For Algorithm 5, the application of some precedence relations is exhibited.

1. For the delimiter symbol \$:
   (a) $\$ \lessdot \mathcal{L}(S) \rightarrow$ Rule `(A.1)`.
   - $\$ \lessdot a$
   - $\$ \lessdot b$
   - $\$ \lessdot d$
   - $\$ \lessdot c$

   (b) $\mathcal{R}(S) \gtrdot \$ \rightarrow$ Rule `(A.2)`.
   - $c \gtrdot \$$

2. For $S \longrightarrow ABC$
   (a) $X = A$, $Y = B \rightarrow$ Rule `(E.1)`.
   $\mathcal{R}(A) \gtrdot \mathcal{L}_m(B)$
   - $a \gtrdot b$

   $u = null$, $l = [A]$
   (b) $X = B$, $Y = C \rightarrow$ Rule `(E.1)`.
   $\mathcal{R}(B) \gtrdot \mathcal{L}_m(C)$
   - $b \gtrdot c$

$A \in l \neq [\,] \rightarrow$ Rule (E.3).
$\mathcal{R}(A) \gtrdot \mathcal{L}_m(C)$
- $a \gtrdot c$

$u = null$, $l = [\,]$

3. For $A \longrightarrow aA$
   (a) $X = a$, $Y = A \rightarrow$ Rule (C).
   $a \lessdot \mathcal{L}(A)$
   - $a \lessdot a$

   $u = a$, $l = [\,]$
4. For $B \longrightarrow bB$
   (a) $X = b$, $Y = B \rightarrow$ Rule (C).
   $b \lessdot \mathcal{L}(B)$
   - $b \lessdot b$

   $u = b$, $l = [\,]$
5. For $C \longrightarrow CDc$
   (a) $X = C$, $Y = D \rightarrow$ Rule (E.1).
   $\mathcal{R}(C) \gtrdot \mathcal{L}_m(D)$
   - $c \gtrdot d$

   $u = null$, $l = [\,]$
   (b) $X = D$, $Y = c \rightarrow$ Rule (D.1).
   $\mathcal{R}(D) \gtrdot c$
   - $d \gtrdot c$

   $u = null$, $l = [\,]$
6. In the following cases, the form XY is not fulfilled in the right side part of the production; therefore, no rule can be applied:
   (a) $A \longrightarrow a$
   (b) $B \longrightarrow b$
   (c) $B \longrightarrow \varepsilon$
   (d) $C \longrightarrow c$
   (e) $D \longrightarrow d$

The remaining precedence relations are derived using the same method as the previously demonstrated cases.

5.3.6. Operator Precedence Table

At the end of the algorithm execution, operator precedence Table 16 is obtained.

Table 16. Precedence Table of Example 3.

	a	b	c	d	\$
a	$\lessdot$	$\gtrdot$	$\gtrdot$		
b		$\lessdot$	$\gtrdot$		
c				$\gtrdot$	$\gtrdot$
d			$\gtrdot$		
\$	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$	

5.3.7. Bottom-Up Parsing

Let *aabbcdc* be a string obtained by rightmost derivation, then applying Algorithm 1 results in the bottom-up parsing detailed in Table 17.

Table 17. Bottom-up Parsing of Example 3.

Stack	Input	Action
\$	*aabbcdc*\$	Shift
\$*a*	*abbcdc*\$	Shift
\$*aa*	*bbcdc*\$	Reduce
\$*a*	*bbcdc*\$	Reduce
\$	*bbcdc*\$	Shift
\$*b*	*bcdc*\$	Shift

Table 17. *Cont.*

Stack	Input	Action
\$*bb*	*cdc*\$	Reduce
\$*b*	*cdc*\$	Reduce
\$	*cdc*\$	Shift
\$*c*	*dc*\$	Reduce
\$	*dc*\$	Shift
\$*d*	*c*\$	Reduce
\$	*c*\$	Shift
\$*c*	\$	Reduce
\$	\$	***Accept***

5.4. Discussion

In the examples, three CFGs are considered: one OPG and two non-OPGs. In the first CFG, it is observed that it complies with the basic rule, while the following ones do not since they present productions with consecutive non-terminal symbols on the right side. Regardless of the nature of the CFG, the proposed algorithms play a crucial role in obtaining the operator precedence table. Using Algorithm 1 applied to a specific string for each CFG, a bottom-up parsing is performed, verifying that the strings are generated by their respective grammars. This process underscores the functionality of the proposal to perform bottom-up parsing based on operator precedence, even when the CFGs do not fulfill the basic requirement of the OPGs.

6. Exceptions

If the CFG has any of the following forms, the rules outlined in the algorithms of this paper would not apply:

1. $A \longrightarrow aBb$
   $B \longrightarrow \beta a$
   The sets for B in the second production are:
   $\mathcal{L}(B) = \{a\}$.
   $\mathcal{R}(B) = \{a\}$.
   The precedence relations based on the sections of Algorithm 5 exhibit:
   - For the rule `(C)`: $a \lessdot a$ and $u = a$.
   - For the rule `(D.1)`: $a \gtrdot b$.
   - For the rule `(D.2)`: $a \doteq b$.

   Applying rules `(D.1)` and `(D.2)` generates the conflict.

2. $A \longrightarrow aBb$
   $B \longrightarrow a\beta$
   Being $\beta \overset{*}{\Longrightarrow} \varepsilon \vee \beta \overset{*}{\Longrightarrow} \gamma a$.
   According to the established conditions, the sets for B in the second production are
   $\mathcal{L}(B) = \{a\}$.
   $\mathcal{R}(B) = \{a\}$.
   The precedence relations based on the sections of Algorithm 5 exhibit:
   - For the rule `(C)`: $a \lessdot a$ and $u = a$.
   - For the rule `(D.1)`: $a \gtrdot b$.
   - For the rule `(D.2)`: $a \doteq b$.

   The application of rules `(D.1)` and `(D.2)` generates the conflict.
   If the $\mathcal{R}(B)$ does not contain a, then the conflict does not arise.
3. $A \longrightarrow bBa$
   $B \longrightarrow a\beta$
   The sets for B in the second production are:
   $\mathcal{L}(B) = \{a\}$.
   $\mathcal{R}(B) = \{a\}$.
   The precedence relations based on the sections of Algorithm 5 exhibit:
   - For the rule `(C)`: $b \lessdot a$ and $u = b$.
   - For the rule `(D.1)`: $a \gtrdot a$.
   - For the rule `(D.2)`: $b \doteq a$.

   The application of rules `(C)` and `(D.2)` generates the conflict.
4. $A \longrightarrow bBa$
   $B \longrightarrow \beta a$
   Being $\beta \overset{*}{\Longrightarrow} \varepsilon \vee \beta \overset{*}{\Longrightarrow} a\gamma$.
   According to the established conditions, the sets for B in the second production are:
   $\mathcal{L}(B) = \{a\}$.
   $\mathcal{R}(B) = \{a\}$.
   The precedence relations based on the sections of Algorithm 5 exhibit:
   - For the rule `(C)`: $b \lessdot a$ and $u = b$.
   - For the rule `(D.1)`: $a \gtrdot a$.
   - For the rule `(D.2)`: $b \doteq a$.

   The application of rules `(C)` and `(D.2)` generates the conflict.
   If the $\mathcal{L}(B)$ does not contain a, then the conflict does not arise.

When CFG productions take the following form, there will be no conflicts:
$A \longrightarrow aBb$
$B \longrightarrow \gamma$
where γ has neither a at the end of the right side nor b at the end of the left side.

7. Conclusions

Previous work has addressed the application of OPGs to solve problems related to the parsing of various forms of languages; however, not much emphasis has been placed on the definition of the Left and Right sets nor on the generation of an algorithm to find the operator precedence table.

In this work, we have redefined the sets Right and Left, previously exposed by other authors, and introduced a novel set called Leftmost. These sets form the basis for an algorithm that constructs the operator precedence table. Each set is constructed from an algorithm that facilitates its obtaining, adding a new dimension to the existing research.

The proposed algorithm for constructing the precedence table, while breaking (with some exceptions) the basic definition of an OPG, opens up new possibilities. It allows for the construction of an operator precedence table from a non-OPG CFG, paving the way for a bottom-up parsing algorithm to recognize a string generated by the CFG.

Three examples of CFGs are shown: one OPG and two non-OPGs. The proposed algorithms are systematically applied to these examples, step by step, until the operator precedence tables are obtained, regardless of the CFG's nature. Additionally, the bottom-up parsing algorithm is applied to specific strings for each grammar, providing verification that they generated them.

It is important to note that while the algorithm and the definition of the new sets of terminal symbols are significant advancements, they do have limitations. It is not always possible to solve all cases where two or more adjacent terminal symbols are on the right-hand side. There are exceptions, and we must be aware of them.

Author Contributions: Conceptualization, L.L. and J.M.; methodology, J.M.; software, L.L.; validation, L.L. and J.M.; formal analysis, L.L., E.A., and J.M.; investigation, L.L. and J.M.; resources, J.M.; data curation, E.A.; writing—original draft preparation, E.A. and J.M.; writing—review and editing, E.A. and J.M.; visualization, E.A.; supervision, J.M.; project administration, J.M.; funding acquisition, J.M.; All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding and the APC was funded by Universidad del Norte.

Data Availability Statement: Data sharing is not applicable to this article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Floyd, R.W. Syntactic analysis and operator precedence. *J. ACM (JACM)* **1963**, *10*, 316–333. [CrossRef]
2. Alfred, V.; Monica, S.; Ravi, S.; Jeffrey, D.U. *Compilers Principles, Techniques*; Pearson Education: London, UK, 2007.
3. Grune, D.; Jacobs, C.J.H. *Parsing Techniques (Monographs in Computer Science)*; Springer: Berlin, Heidelberg, 2006.
4. Crespi-Reghizzi, S. An effective model for grammar inference. In *Information Processing 71*; Gilchrist, B., Ed.; Elsevier: North-Holland, New York, 1972; pp. 524–529.
5. Crespi-Reghizzi, S. Reduction of Enumeration in Grammar Acquisition. In Proceedings of the 2nd International Joint Conference on Artificial Intelligence, San Francisco, CA, USA, 1–3 September 1971; pp. 546–552.
6. Crespi-Reghizzi, S.; Melkanoff, M.A.; Lichten, L. The Use of Grammatical Inference for Designing Programming Languages. *Commun. ACM* **1973**, *16*, 83–90. [CrossRef]
7. von Braunmühl, B.; Verbeek, R. Input Driven Languages are Recognized in log n Space. In *Topics in the Theory of Computation*; North-Holland Mathematics Studies; Karplnski, M., van Leeuwen, J., Eds.; Elsevier: North-Holland, The Netherlands, 1985; Volume 102, pp. 1–19.
8. Mehlhorn, K. Pebbling mountain ranges and its application to DCFL-recognition. In *Automata, Languages and Programming*; de Bakker, J., van Leeuwen, J., Eds.; Springer: Berlin/Heidelberg, Germany, 1980; pp. 422–435.
9. Mandrioli, D.; Pradella, M. Generalizing input-driven languages: Theoretical and practical benefits. *Comput. Sci. Rev.* **2018**, *27*, 61–87. [CrossRef]
10. Alur, R.; Madhusudan, P. Adding Nesting Structure to Words. *J. ACM* **2009**, *56*, 1–43. [CrossRef]
11. Crespi Reghizzi, S.; Mandrioli, D. Operator precedence and the visibly pushdown property. *J. Comput. Syst. Sci.* **2012**, *78*, 1837–1867. [CrossRef]
12. Crespi Reghizzi, S.; Pradella, M. Beyond operator-precedence grammars and languages. *J. Comput. Syst. Sci.* **2020**, *113*, 18–41. [CrossRef]
13. Barenghi, A.; Crespi Reghizzi, S.; Mandrioli, D.; Panella, F.; Pradella, M. Parallel parsing made practical. *Sci. Comput. Program.* **2015**, *112*, 195–226. [CrossRef]
14. Crespi-Reghizzi, S.; Mandrioli, D.; Martin, D.F. Algebraic properties of operator precedence languages. *Inf. Control* **1978**, *37*, 115–133. [CrossRef]
15. Chomsky, N. Three models for the description of language. *IRE Trans. Inf. Theory* **1956**, *2*, 113–124. [CrossRef]
16. Aho, A.V.; Ullman, J.D. *The Theory of Parsing, Translation, and Compiling*; Prentice-Hall Englewood: Cliffs, NJ, USA, 1973; Volume 1.
17. Louden Kenneth, C. Compiler Construction: Principles and Practice. In *Course Technology*; PWS Publishing Co.: Montgomery, IL, USA, 1997.