ISSN : 0976-8491 (Online) | ISSN : 2229-4333 (Print)

Ambiguity Detection Algorithm for Context free Grammar

[1]Saurabh Kumar Jain, [2]Ajay Kumar
[1,2]Dept. of CSE, Thapar University, Patiala, Punjab, India

Abstract
One way to verifying a grammar is the detection of ambiguity. Unfortunately, ambiguity problem for context-free grammars is undecidable. Ambiguity in context-free grammars is a recurring problem in language design and parser generation, as well as in applications where grammars are used as models of real-world physical structures. Context-free grammars are widely used but still hindered by ambiguity.
We observe that there is simple linguistic characterization of the grammar ambiguity problem .This problem divided into form of horizontal and vertical ambiguity. We show the conservative approximation for ambiguity problem. The proposed methodology have implemented in Java.

Keywords
Ambiguity, Context Free Grammar, Horizontal Ambiguity, Vertical Ambiguity.

I. Introduction
When describe the context free grammar, aware of ambiguity. Ambiguity means generate the multiple parse tree for the single string. Context free grammar have many important application, for example in the field of bioinformatics where used for sequence comparison and RNA secondary structure analysis. Several important algorithms such as Viterbi algorithm on stochastic CFGs give the incorrect results in presence of ambiguity. There are many algorithms have been proposed for ambiguity detection but they are not applicable for every grammar.
Saul Gorn [4] described a Turing Machine to generate all possible strings of a grammar. Searching (string generated before or not generated) starts after generation of each new string. If the string exists before then the string has multiple derivations, the grammar is ambiguous. The searching process for the string is a simple Breadth First Search.
Cheung et al [5] give a searching method with pruning of all possible derivations of a grammar. This method is nothing but an optimization of method described by Saul Gorn. This method can also detect non-ambiguity for some languages with an infinite language.
Schroer [7] developed an ambiguity detection tool "AMBER". It uses Earley parser to generate all possible strings and then finds duplicates in them. The way of derivation of a string of a grammar is similar to the methodology given by Saul Gorn except for some variations in the search method which helps in improving the search timing and comparing the strings. LR (k) parsing algorithms described by Knuth [9] makes decisions based on k-input symbols of look-ahead. It is based on the bottom-up parsing technique. LR (k) grammars can be deterministically parsed using this algorithm. For applying this method, a parse table has to be maintained which helps in identifying the action (shift, reduce, accept or error) to be performed. LR (k) testing is very simple for detecting ambiguity: if the parse table contains no conflicts then the grammar is LR (k). Similarly, the test can be used for ambiguity detection.
Brabrand et al[10] presented a new approach for detecting ambiguity. They divided the ambiguity detection problem into two sets of problems of similar types called as Vertical and Horizontal ambiguity. In this method, languages of the productions are approximated to make the intersection and overlap problem decidable. By approximating, we can represent the language into regular grammar; this process is called as conservative approximation. By doing this we can compute the regular approximations of vertical and horizontal ambiguity.
In our approach we present the methodology for computing the horizontal and vertical ambiguity after converting the grammar into CNF.

II. Definitions
These definitions related to the grammar has been discussed

A. Context-Free Grammars
A grammar for a language is a set of rules that govern the generation of sentences in that language. Every language, whether it is a natural language (English, Spanish, etc.) or a programming language (C++, Java, etc.) has a grammar. Formally, a grammar G can be viewed as a 4-tuple, (V, T, P, S), where V is the set of variables, also called non terminals or syntactic categories, each of which represents a set of strings, T is a finite set of symbols, also called terminals, that form the strings of the language being defined, S is the start symbol that represents the language being defined, and P is the finite set of productions or rules that represent the recursive definition of the language[8].
A production is basically a re-writing rule that consists of a head or left-hand side which is a string of at least one non terminal and zero or more terminals that is being defined, the production symbol '→', and a body or right-hand side which is a string of zero or more terminals and non terminals [2]. The derivation of any string in the language starts from the start symbol. All intermediate stages of the strings resulting from the start symbol in the derivation process are called sentential forms. The derivation of a string can also be represented in the form of a tree called a parse tree or a derivation tree.
Definition 2.1: A grammar is said to be context free if all production in P have the form

$$A \rightarrow x$$

Where $A \in V$ and x $\in (V \cup T)^*$
There are many kinds of normal forms are establish for context free grammar, we are considering only the Chomsky Normal Form for the Grammar(CNF).
Definition 2.2: A context free grammar in Chomsky normal form if all production is of the form
$A \rightarrow BC$ OR $A \rightarrow a$
Where A, B, C $\in$ V and a $\in$ T
Definition 2.3: A context-free grammar G is said to be ambiguous if there exists some that has at least two distinct derivation trees. Alternatively, ambiguity implies the existence of two or more leftmost or rightmost derivations
Definition 2.4: There are basically two types of ambiguity exists in grammar
- Vertical Ambiguity
- Horizontal Ambiguity

Definition 2.5: Vertical ambiguity means that, during parsing of a string, there is a choice between two different productions of a non terminal [10].
For a given grammar G, Two sentential form α and α'| are vertically ambiguous.

$$L_G(\alpha) \cap L_G(\alpha') \neq \Phi$$

Example 2.5

$$C \rightarrow Ay \mid xB$$

$$A \rightarrow xa$$

$$B \rightarrow ay$$

For taking any string xay can be parsed by using the first or the second production of C, so that vertical ambiguity exists
Definition 2.6: Horizontal ambiguity then means that, when parsing a string according to a production, there is choice of how to split the string into substrings corresponding to the entities in the production[10].
For a given grammar G, two sentential form α and α'| are horizontally ambiguous if

$$L_G(\alpha) \overset{\frown\frown}{\cap} L_G(\alpha') \neq \Phi$$

Example 2.6

$$C \rightarrow xAB$$

$$A \rightarrow a \mid \in$$

$$B \rightarrow ay \mid y$$

Also here string xay can be parsed either in xA(using the First production of A and Second of B)or in B(using the second production of A and First Production of B),so that horizontal ambiguity exists.

III. Proposed Approach for Ambiguity Detection

These are the following steps are taken for ambiguity detection in the grammar.

- Convert given grammar into Chomsky Normal Form.
- Computation of first and last function for production P.
- Identify the productions which have a possibility of Vertical and Horizontal ambiguity.
- Check the Vertical and Horizontal Ambiguity for the productions.

A. Chomsky Normal form of the Given Grammar

For the converting grammar into Chomsky Normal form these the following steps are followed

A. Removing Useless Variables Unreachable from the Start Symbol

Those variables which are not used for generation of any string form start symbol are removed.
Given a grammar G = (V,T ,P, S) we would like to remove all variables that are not derivable from S. To this end, consider the following algorithm
Algorithm 3.1: ComReachableVars(V, T, P, S)
Input: A grammar G
Output: Productions which are not reachable from start symbol S.

```
$V_{old} \rightarrow \phi$
  $V_{new} \leftarrow \{S\}$
  While $V_{old} = V_{new}$ do
      $V_{old} \leftarrow V_{new}$
      for $X \in V_{old}$ do
          for (X $\rightarrow$ w) $\in$ P do
              Add all Variables appearing in w to $V_{new}$
 return $V_{new}$
```

2. Removing Useless Variables That do Not Generate Anything

In this step we remove variables that do not generate any string. Given a grammar G = (V,T ,P,S) we would like to remove all variables that are not derivable from S. To this end, consider the following algorithm.
Algorithm 3.2 :CompGeneratingVars(V, T, P, S)
Input: A grammar G
Output: Variables which generates nothing.

```
$V_{old} \rightarrow \phi$
$V_{new} \leftarrow V_{old}$
do
    $V_{old} \leftarrow V_{new}$
    for $X \in V$ do
        for $(X \rightarrow w) \in P$) do
            if $w \in (T \cup V_{old})^*$ then
                $V_{new} \leftarrow V_{new} \cup \{X\}$
while $(V_{old} \neq V_{new})$
return $V_{new}$
```

3. Compute Null Production

Given a grammar G=(V,T,P,S) then the production V$\rightarrow T$ is generate nullable ,or there is way to generate the null string, then apply the following algorithms
Algorithm 3.3: CompNullable(V, T, P, S)
Input: A grammar G
Output: Productions which generates null productions.

```
$V_{null} \leftarrow \phi$
do
  $V_{old} \leftarrow V_{null}$
  for $X \in V$ do
      for $(X \rightarrow w) \in P$ do
          if $w = \in$ or $w \in (V_{null})^*$ then
              $V_{null} \leftarrow V_{null} \cup \{X\}$
while($V_{null} \neq V_{old}$)
return $V_{null}$
```

4. Compute and Remove Unit Production

A unit production is form of X$\rightarrow Y$ where X,Y $\in V$,so that first we have to compute these production and then remove these productions
Algorithm 3.4: CompUnitPairs(V, T, P, S)
Input: A grammar G
Output:Productions which generates Unit Productions.

```
$P_{new} \leftarrow \{(X \leftarrow Y) \mid (X \rightarrow Y) \in P\}$
do
    $P_{old} \leftarrow P_{new}$
    for $(X \rightarrow Y) \in P_{new}$ do
        for $(Y \rightarrow Z) \in P_{new}$ do
            $P_{new} \leftarrow P_{new} \cup \{X \rightarrow Z\}$
    return $P_{new}$
```

Algorithms 3.5: Remove Unit Rules(V, T, P, S)
Input:
Output:

```
$U \rightarrow CompUnitPairs(G)$
$P \leftarrow P \setminus U$
for $(X \rightarrow A) \in U$ do
    for $(A \rightarrow w) \in P_{old}$
        $P \leftarrow P \cup \{X \rightarrow w\}$
return(V,T,P,S)
```

5. Making Production With Two Variables on the Right Side

we might have a rule in the grammar of the form

$X \rightarrow B_1B_2B_3.........B_K$

To make this into a binary rule (with only two variables on the right side, we remove this rule from the grammar, and replace it by the following set of rules

$X \rightarrow B_1Z_1$

$Z_1 \rightarrow B_2Z_2$

$Z_{K-3} \rightarrow B_{K-2}Z_{K-2}$

$Z_{K-2} \rightarrow B_{K-1}B_K$

Example 3.1 Converting Context free Grammar into CNF
Given Grammar G..

$S \rightarrow ASA \mid aB$

$A \rightarrow B \mid S$

$B \rightarrow b \mid \in$

Following steps are used for conversion of CFG to CNF
Step 1 Removing unreachable production from the start symbol
For the given grammar there is no production which is unreachable from the start symbol S

Step 2 Compute and Remove the null production from the grammar.
For the given grammar there exist null production
$B \rightarrow \in$ and replace B with A
$A \rightarrow \in$
So that following grammar generated

$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$

$A \rightarrow B \mid S$

$B \rightarrow b$

Step 3-Compute and Remove Unit Production
The unit pair are {A$\rightarrow B$,A$\rightarrow S$ }so need to remove them from the grammar Generated grammar is

$S \rightarrow ASA \mid aS \mid a \mid SA \mid AS$

$A \rightarrow b \mid ASA \mid Ab \mid a \mid SA \mid AS$

$B \rightarrow b$

Step 4. Making Production with two variables on the right side
These production which consists of more than two variable on right hand side such as(S $\rightarrow$ASA and A$\rightarrow$ASA),make them into maximum two variables.
So final restructured grammar which is in CNF form
$S \rightarrow AZ \mid UB \mid a \mid SA \mid AS$
$A \rightarrow b \mid AZ \mid UB \mid a \mid SA \mid AS$
$B \rightarrow b$
$U \rightarrow a$
$Z \rightarrow SA$

B. Calculation of First and Last Function

These are the Data structures used in the computation of First and Last function.
Set-An unordered collection of distinct objects.
List- ordered collection of objects.
Iterator-That enumerates the content of a set or list.
We represent a production by its LHS and a list of the symbols on its RHS and these procedure used for the algorithms.
PRODUCTIONS( ): Returns an iterator that visits each production in the grammar.
NONTERMINAL(V): Adds V to the set of non terminals. An error occurs if V is already a terminal symbol. The function returns a descriptor for the non terminal.
TERMINAL(T): Adds T to the set of terminals. An error occurs if T is already a Non terminal symbol. The function returns a descriptor for the terminal.
NONTERMINALS( ): Returns an iterator for the set of non terminal
ISTERMINAL(T): Returns true if T is a terminal; otherwise, returns false.
RHS(P): Returns an iterator for the symbols on the RHS of production P
LHS(P): Returns the non terminal defined by production P
PRODUCTIONSFOR(V): Returns an iterator that visits each production for non terminal V.
OCCURRENCES(X): Returns an iterator that visits each occurrence of X in the RHS of all rules.

1. Calculation of First function

First Function

This is set of all terminal symbols that can begin a production. First function are used after scanning from left to right of the production
Algorithm 3.5
Input- Grammar G
Output- First functions corresponding to Grammar G.
Function First(α):Set

```
    For each A $\in$ Nonterminal() do Visited First(A) $\leftarrow$ false

    First $\leftarrow$ InternalFirst()
    return (First)
end

function InternalFirst(Xβ):Set
```

```
  if Xβ = empty
  then return (ϕ)
  if X =V
  then return ({X})
  X is a nonterminal.
  First ← ϕ
    if not VisitedFirst(X)
 then
    VisitedFirst(X)
     For each rhs ∈ ProductionsFor(X) do
       First ← First ∪ InternalFirst(rhs)
   if SymbolDerivesEmpty(X)
   then First ← First ∪ InternalFirst(β)
return (ans)
```

First(α) is computed by invoking FIRST(α). Before any sets are computed, VisitedFirst(X) for each nonterminal A. VisitedFirst(X) is to indicate that the productions of X already participate in the computation of First(α).

2. Calculation of Last Function

Reverse the production and then calculate the first function which is the last function for the given productions

C. Checking Vertical Ambiguity Productions

For the given Grammar G', which is now in CNF. First of all check all the productions which have the possibility of ambiguity. All productions have been checked with function CheckProduction() and then return the LHS value which consists of nonterminals.

```
Algorithm 3.6
Input: Grammar G containing production $P_1$, $P_2$----------$P_i$
Output: Production P ∈ $P_i$ containing Vertical Ambiguity
Procedure CheckProduction()
   For each productions ()
     Visited LHS(P) ← true
     Count RHS(P)>1
Return LHS(P)

Procedure CheckVambiguity()
   For each LHS(P) containing type production for non terminal
A → $P_1$|$P_2$
        RHS(P) ∈ FirstLast($P_1$,$P_2$)
     CALL [FirstLast($P_1$)]
     CALL [FirstLast($P_2$)]
If First,Last(P1) ∩ First,Last(P2) ≠ ϕ
LHS(P) contain vertical ambiguity
```

D. Checking Horizontal Ambiguity Production

```
Algorithm 3.7:
Input: Grammar G containing productions $P_1$,$P_2$--------$P_i$
Output: Productions P ∈ Pi containing Horizontal Ambiguity

Procedure CheckProduction()
For each production ()
Visited LHS(P) ← true
Corresponding RHS(P) ← true
RHS(P) ∈ [Nonterminal(P) ≥ 1]
   Production(Nonterminal(P)>1)
Return LHS(P)

Procedure  CheckHambiguity()
For each production() containing type A → $P_1$$P_2$
    CallNonterminal()
    For each Nonterminal($P_1$,$P_2$)
          Call First($P_1$,$P_2$)
          Call last($P_1$,$P_2$)
         FirstLast($P_1$)=First($P_1$) ∪ Last($P_1$)
         FirstLast($P_2$)=First($P_1$) ∪ Last($P_2$)
         Production($P_1$)h= FirstLast($P_1$)
         Production($P_2$)h=FirstLast($P_2$)
If [Production(P1)h ∩ Production(P2)h] ≠ ϕ
LHS (P) contain Horizontal Ambiguity
```

Example 3.2
Consider the following CFG

$$C \to Ay$$
$$C \to xB$$
$$A \to xa$$
$$B \to ay$$

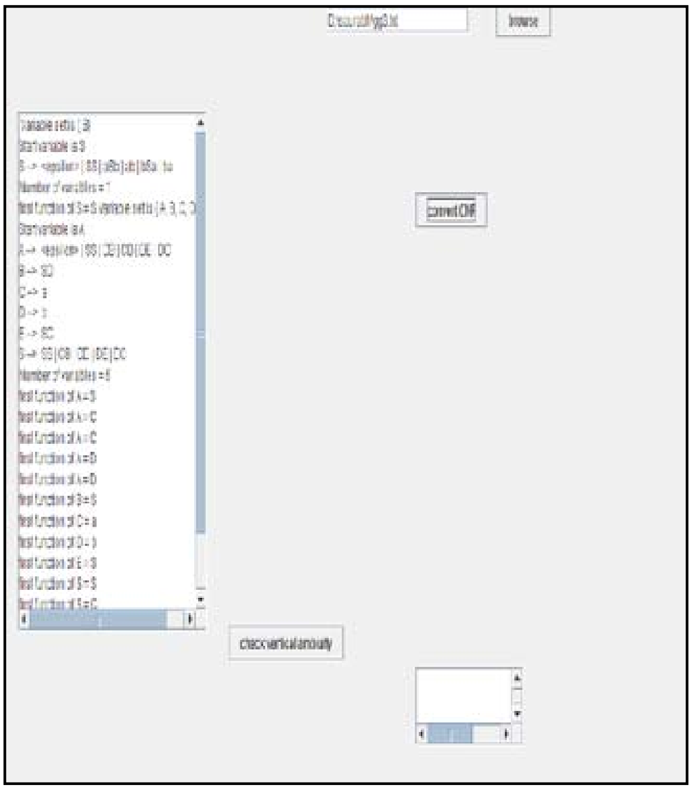
After applying the algorithm for CFG to CNF conversion,we get

$$C \to AD$$
$$C \to EB$$
$$A \to EF$$
$$E \to x$$
$$B \to FD$$
$$F \to a$$
$$D \to y$$

Checking vertical ambiguity
RHS(P)>1 true for production $C \to AD \mid EB$
LHS(P)= C
Call(First(AD))={x}
Last(AD)={y}
FirstLast(AD)={x,y}
Call(First(EB))={x}
Last(EB)={a}
FirstLast(EB)={x,a}
FirstLast(Ay) ∩ FirstLast(xB) ≠ Φ
So this grammar contain vertical Ambiguity

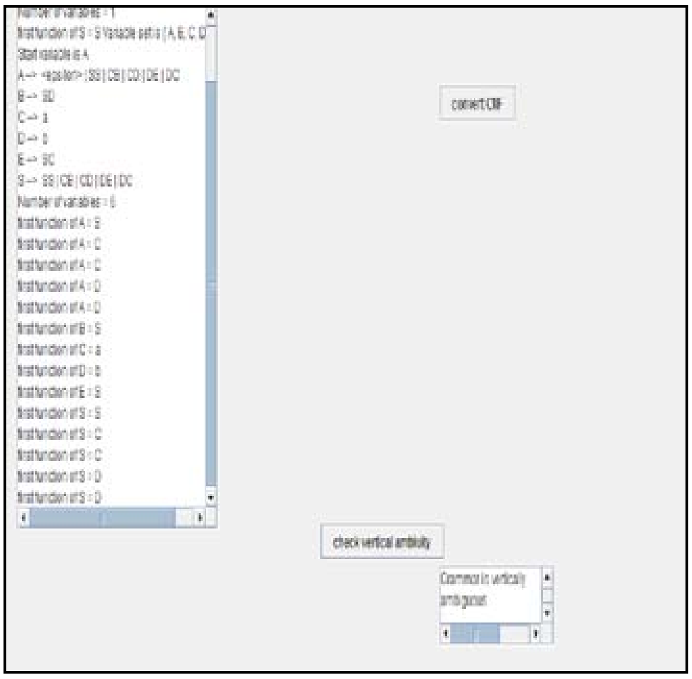
IV. Implementation

This section is given for the implementation of the proposed algorithms on various grammars. The proposed algorithms have been applied on set of grammar after converting into CNF form. For the test we include the grammars of different sizes, ambiguous grammars which contain Horizontal, Vertical Ambiguity or Unambiguous grammar.



```
Variable set is {S}
Start variable is S
S -> <epsilon> | SS | aSb | ab | bSa | ba
Number of variables = 1
first function of S = S Variable set is {A, B, C, D
Start variable is A
A -> <epsilon> | SS | CB | CD | DE | DC
B -> SD
C -> a
D -> b
E -> SC
S -> SS | CB | CD | DE | DC
Number of variables = 6
first function of A = S
first function of A = C
first function of A = C
first function of A = D
first function of A = D
first function of B = S
first function of C = a
first function of D = b
first function of E = S
first function of S = S
first function of S = C
```

Fig. 1: CNF Conversion of CFG



```
Number of variables = 1
first function of S = S Variable set is {A, B, C, D
Start variable is A
A -> <epsilon> | SS | CB | CD | DE | DC
B -> SD
C -> a
D -> b
E -> SC
S -> SS | CB | CD | DE | DC
Number of variables = 6
first function of A = S
first function of A = C
first function of A = C
first function of A = D
first function of A = D
first function of B = S
first function of C = a
first function of D = b
first function of E = S
first function of S = S
first function of S = C
first function of S = C
first function of S = D
first function of S = D
```

```
Grammar is vertically
ambiguous
```

Fig. 2: Detection of Horizontal and Vertical Ambiguity of CNF grammar.

V. Conclusion and Future Scope

The presented algorithms implemented in Java, identify the horizontal and vertical ambiguity in the context free grammar. It converted into CNF because these form easily implemented for the parsing and beneficial in terms of computation. The presented algorithm is less complex than other approaches in the literature.

References

[1] M. Kruse, "Ambiguity Detection for Context-Free Grammars in Eli", Bechlor's Thesis, University of Paderborn: Germany, 2008.

[2] J.E Hopcroft, R.Motwani, J.D.Ullman,"Introduction to Automata Theory, Languages, and Computation", Pearson Education Asia, Delhi, India.

[3] H.J.S Bastern,"Ambiguity Detection Methods for Context-Free Grammars", Mater's Thesis, University of Amsterdam: The Netherlands.

[4] S.Gorn,"Detection of Generative Ambiguities in Context-Free Mechanical Languages", JACM, Vol no. 10(2), pp. 196-208,1963.

[5] B.S.N Cheung, R.C Uzgalis,"Ambiguity in Context-Free Grammars", SAC'95, Proceedings of the 1995 ACM Symposium on Applied Computing, 272-276, 1995.

[6] B.S.N Cheung,"A Theory of Automatic Language Acquisition", Ph.D Thesis, University of Hong Kong: China, 1994.

[7] F.W.Schroer, "AMBER: An Ambiguity Checker for Context-Free Grammar", Technical report.

[8] S.Jampana, "Exploring the Problem of Ambiguity in Context-Free Grammar", Master's Thesis, Oklahoma State University, 2005.

[9] D.E.Kruth, "On the Translation of Languages from Left to Right", Information and Control, Vol. 8(6), pp. 607-639, 1965.

[10] C.Brabrand, R. Giegerich, A. Møller, "Analyzing Ambiguity of Context Free Grammar", Proc. 12th International Conference on Implementation and Application of Automata, CIAA '07, 2007.

Saurabh Jain has completed Btech from B.B.D.I.T Ghaziabad. He has teaching experience of three years. Currently he is pursuing Master of Engineering in Software Engineering. His area of interest is Theoretical Computer Science.

Ajay Kumar Loura as an Assistant Professor in Thapar University, Patiala, India. He has nine years of teaching experience in the area of Theory of Computation, software teting and programming Languages. His research interests are Theoretical Computer Science and Software testing.