ABSTRACTION OF GRAMMAR AND PARSE TREE FROM THE INPUT STRING

Dr.R N Kulkarni
Dept. of CSE
BITM Bellary
m_kulkarni@rediffmail.com

Arpitha D
Dept. of CSE
BITM Bellary
dsl.arpitha@gmail.com

Ashwini B Shiraguppi
Dept. of CSE
BITM Bellary
ashubs005@gmail.com

Mustour Shruthi
Dept. of CSE
BITM Bellary
logon2shruthi@gmail.com

Abstract :

In the field of compiler design grammars plays a vital role. The grammar allows us to write a computer program to determine whether a string of statement is syntactically correct in the programming language or not. Different authors feels that natural language such as English could be analyzed as precisely, because the programs what we write consists of English statements. The use of Context Free Grammars (CFGs) for syntax definition is increased as it is used for verifying the syntax of computer programming language.In this paper, we are proposing a automated methodology for the abstraction the required string from the input and then draw parse tree for the generated string.

Keywords: Context Free Grammar, ambiguous, Parse tree.

1. INTRODUCTION:

In the field of compiler design grammars play a vital role. The grammar allows us to write a computer program to determine whether a string of statements is syntactically correct in the programming language. Many people would wish that natural languages such as English could be analyzed as precisely, that we could write computer programs to tell which English sentences are grammatically correct.The use of Context Free Grammars (CFGs) for syntax definition is already widespread and keeps growing. Primarily, Context Free Grammar or CFG used to build compilers to verify the syntax of computer programming language. There is no tool available where we can extract the parse tree and unambiguous grammar by giving the grammar and string.

1.1 Literature Survey

The paper [1] discusses about the way of verifying a grammar is the detection of ambiguities. Author discuss about a different methods of testing the ambiguity of the grammar: the derivation generator AMBER, the LR(k) test and the Non-canonical Un-ambiguity test. The proposed tool in [1] is referred to as derivation generator which derives the derivation sequence. The paper [2] discuss about the language model presented by Comosky, degree of ambiguity and comparisons of existing methods and recent trends.

The paper [3] discusses the new algorithm to detect ambiguity in character level grammar. The new method showed that the time taken by the ambiguity detection algorithm for character level

grammar for languages such as C and Java is significantly reduced without any loss.

The paper [4] discusses about the tool that pinpoints the possible ambiguities in Context Free Grammars and this tool implements a conservative algorithm [4] which guarantees no ambiguity will be overlooked.

The paper [5] discuss about grammar which allows us to write a computer program to determine whether a string of statements is syntactically correct in the programming language

The authors in [6, 7, 8] discuss about basic ideas of grammars and formal properties of Context-Free Grammar, techniques to resolve ambiguous grammar and introduces many of important concept about ambiguous and unambiguous grammar, parse tree, illustrates the key theoretical concepts of Compilers and its phases. The basic Notations and concepts of Grammars and languages [8] gives detailed description of syntax-directed translation using LL(1) Grammars. The link [9] discuss about the pictorial representation of the parse tree.

In our proposed methodology we are trying to identify first whether the given grammar is ambiguous or unambiguous. If it is ambiguous then we convert it to unambiguous first and then abstracts the required string and Generate the parse tree by applying production rules from the given Grammar.

2. TERMINOLOGY

2.1 Context free grammar: Context Free Grammars are widely used for describing formal languages including programming languages. The CFGs includes ambiguous grammars-those which can parse inputs in more than one way. Context-free grammars are simple enough to allow the construction of efficient parsing algorithms which, for a given string, determine whether and how it can be generated from the grammar [6].

2.2 Ambiguity: 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. Ambiguities are very hard to detect by hand, so automated ambiguity checkers are welcome tools [3].

2.3 Parse tree: The parse tree is a concrete representation of the input. A concrete syntax tree or parse tree or parsing tree is an ordered, rooted tree that represents the syntactic structure of a string, according to some context-free grammar. Parse trees are usually constructed according to one of two competing relations [9].

3. PROPOSED METHODOLOGY:

Proposed system explores Problem of ambiguity, Approach to Detect Ambiguity and deals with what Ambiguous Grammar is and How to fix them and then generate parse tree.

3.1 Ambiguity checking:

If grammar generates more than one parse tree for the same string then the given grammar is ambiguous.

3.1.1 Algorithm for checking ambiguity

Input: Grammar, String.
Output: Display the message ambiguous or unambiguous.
Notations: N, T, P , S.
N ‧ Set of Non-terminal Symbols represented in upper case letters.
T ‧ Set of terminal Symbols represented in lower case letters.
P ‧ productions of grammar consists of :
(a) A non-terminal called head or left side of the production.
(b) The symbol ‧.

(c) A body or right side consisting of zero or more terminals and non-terminals.
S ・Start symbol.

Step1: [Read Input]
Enter the input for N, T, P, S and string.
Step2: [Deriving string]
From the Start symbol(S) derive the string by using leftmost derivation.
Using Leftmost derivation replace the Start symbol with one of its Production body.
Step3: Repeat step2 until the string is derived.
Step4: [Checking ambiguity]
If more than one derivation sequence generated for the same input string
Then
Display given grammar is ambiguous
Otherwise
Display given grammar is unambiguous.
Step5: End.

3.2 Conversion from ambiguous to unambiguous

If the given grammar is ambiguous then convert it into unambiguous grammar.

3.2.1 Algorithm for Converting Ambiguous to Unambiguous

Input: Ambiguous grammar
Output: Unambiguous grammar.
Notations: E, T, F, id.
E ・ Start symbol
T ・Term
F ・Factor
Id・Identifiers
Step1: [Read the ambiguous grammar]
E->E+E|E*E|(E)|id
Step2: To eliminate ambiguity by rewriting the Grammar.
By Enforces precedence of * over + and Enforces left associativity of + and *
Step3: E represents expressions consisting of terms separated by + signs, T represents terms consisting of factors separated by * signs and F represents factors that can be either parenthesized expressions or identifiers:
E->E+T|T
T->T*F|F
F->(E)|id
Step4: End.

3.3 Parse tree generation

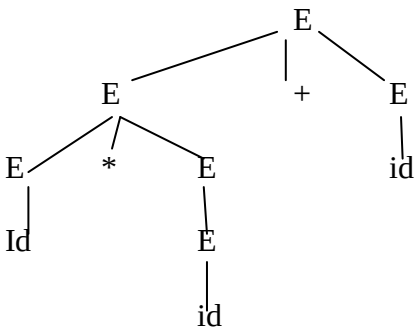
If the given grammar is unambiguous then it generates the parse tree.

3.3.1 Algorithm for generating parse tree.

Input: String, Derivation sequence.
Output: Parse tree.
Notations: E, id.
E・Start symbol
Id ・Identifiers
Step1:[Read Unambiguous grammar]
Take resulting grammar N, T, P, S and string.
Step2: [Constructing parse tree]
Start symbol is labeled as root (E).
The next level of the parse tree can be placed by one of its Production body.
The interior nodes of parse tree are labeled by non-terminals.
The leaf nodes of parse tree are labeled by terminals.
Yield of Parse tree is read from left to right
Step3: End.

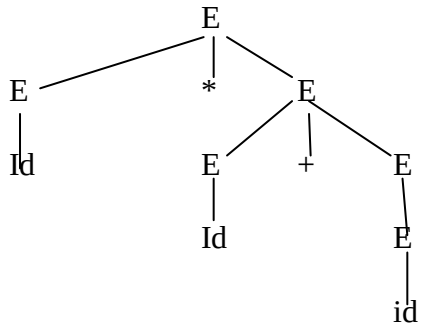
4. CASE STUDY:

Algorithm:
Name: ambiguity checking
Input: grammar, string
Output: display the message ambiguous or unambiguous.
Function:
Grammar: E->E+E|E*E|(E)|id
String: id * id + id.
Derivation1:
E->E + E
->E * E + E
->id * E+ E
->id * id+ E
->id * id + id



Parse tree yields: id*id +id

Derivation2:
E->E * E
->id * E
->id * E + E
->id * id + E
->id * id + id



Parse tree yields: id*id +id

5. CONCLUSION:

In this paper, we proposed an automated methodology that abstracts the parse tree from the given grammar. The proposed methodology is carried out by applying a series of steps to the input string. The methodology is tested for its correctness and completeness.

6. REFERENCE:

[1] The Usability of Ambiguity Detection Methods for Context-Free Grammars by H.J.S. Basten. Electronic Notes in Theoretical Computer Science 238(2009) 35-46.

[2] Advances in Ambiguity Detection Methods for Formal Grammars by Hari Mohan Pandey. In International Conference on Advances in Engineering © 2011 Published by Elsevier Ltd. Selection and/or peer-review under responsibility of ICAE 2011.

[3] Ambiguity detection:scaling to scannerless by H.J.S.Basten,P.kint and J.J.Vinju, pre-proceedings of the 4th International conference on software language engineering, Braga, Portugal, july-2011.

[4] An Experimental Ambiguity Detection Tool by Sylvain Schmitz at Laboratoire I3S Electronic Notes in Theoretical Computer Science .203(2008)69-84.

[5] Formal Grammars and Languages by Tao Jiang, Ming Li, Bala Ravikumar and Kenneth W. Regan Department of Computer Science McMaster University Hamilton, Ontario L8S 4K1, Canada

[6] *Compiler Construction Principles & Practice* by Kennetch C Louden .International student edition Published by Vikas publishing house.

[7] *Compilers: Principles, Techniques and Tools* by Alfred V Aho, Ravi Sethi, Jeffrey D. Ullman second edition Published by Dorling Kindersley(india) Pvt. Ltd licensees of Pearson Education 2009.

[8] The theory and Practice of complier writing International Edition 1985 by Jean-Paul Tremblay, Paul G.Sorenson published by McGraw-Hill ISBN-0-07-065161-2.

[9] Compilers CMPT 379 by Anoop Sarka http://www.cs.sfu.ca/~anoop