Research Journal of Computer and Information Technology Sciences _____ E-ISSN 2320 – 6527
Vol. **4(3),** 1-3, March **(2016)** *Res. J. Computer and IT Sci.*

A Study and Analysis of Precedence Functions for Operator Precedence Parser in Compiler Design

Priyanka Agiwal[1*], Sarita Sharma[1], Kailash Chandra[2], Shivlal Mewada[3] and Pradeep Sharma[1]
[1]Department of Computer Science, Holkar Science College, DAVV, Indore, India
[2]State Forensic Science Laboratory, Home (Police) Dept., Govt. of MP, Sagar, India
[3]Department of Computer Science, MGCGV, Chitrakoot, Satna, India
priyankaagiwal15@gmail.com

Available online at: www.isca.in, www.isca.me
Received 29th July 2015, revised 23th January 2016, accepted 7th March 2016

Abstract

The performance of a computer system rely on compiler technology which states that compiler is used as a mechanism in evaluating architectural approaches before a computer is manufactured. In compiler the parser obtains a string of symbols from the lexical analyzer and ascertains that the string of token names can be accepted by the grammar for the source language. Operator precedence parsing is implementation of shift reduce grammar. In this paper we have studied problem found in the operator precedence relation and precedence relation table. It takes a lot of space in memory to parse a given string. We try to design an algorithm by which one can construct a directed graph and derive the precedence function table for context free grammar by which less memory space is enough for parsing an input string and to motivate the researcher, while showing the future aspects in the area of compiler designing and error solving.

Keywords: Compiler, Parser, Context Free Gramma.

Introduction

A compiler is a set of programs that transforms source code written in a programming language into another computer language. A parser is a compiler or interpreter component that breaks data into smaller elements for easy translation into another language. The shift reduce operator precedence parsing is conceptually very simple and very effective technique for syntactical analysis [1]. Shift reduce parsing is a form of bottom up parsing in which a stack holds grammar symbols and an input buffer holds the rest of string to be parsed. Bottom up parsers for a large class of context free grammar can be easily developed using operator precedence grammar.

According to Yang the parsing process is considered as the manipulation of binary operations of words [2]. Parsing is a fundamental support to improve the performance of natural language processing applications such as machine translation and information retrieval [3].

An operator precedence grammar is a kind of grammar for formal languages. Regular languages are inadequate for specifying all but the simplest aspects of programming language syntax[4]. To specify more complex languages use context free grammar such as:
$L= \{w \in \{a,b\}^* \mid w=a^nb^n \text{ for some n }\}$,
$L=\{w \in \{(,)\}^* \mid$ w is a well-balanced string of parentheses$\}$,
The syntax of most programming languages.

A context free grammar consists of terminals, non-terminals, a start symbol and productions. Terminals are the basic symbols from which strings are formed. Non-terminals are syntactic variables that denote the set of strings. In a grammar one non-terminal is distinguished as the start symbol and the set of strings it denotes is the language generated by the grammar. The production of grammar specifies the manner in which the terminals and non-terminals can be combined to form strings [5].

An operator precedence grammar is a context free grammar that has the property that no production has ε or two adjacent non-terminals in its right hand side [6].
Example- E -> EAE/id
A-> +/-/*

The above example is not an operator precedence parser. The operator precedence parsers usually do not store the precedence table with the relations rather they are implemented in a special way. Operator precedence scheme has been suggested as a means for constructing parsing algorithms for a large class of languages. Operator precedence algorithms need smaller tables, and are very fast in a parse, they lose some semantic information by disregarding non-terminal symbols [7].

The rest of this paper is organized as follows: In section II, we describe about the precedence relation between terminals. In section III, we construct a precedence relation table for terminals. In section IV, we define an algorithm for precedence functions and finally in section V, we conclude my paper and future work.

Precedence Relation: Precedence relations guide the selection of handles and have the following meaning:

Table-1
Precedence Relation Meaning[8,9]

Relation	Meaning
x=◦ y	x "has same precedence as" y
x<◦ y	x "has yields precedence to" y
x ◦> y	x "has takes precedence over" y

Rules for obtaining precedence relation between terminal symbols of a grammar:

x=◦ y – "x has same precedence as y" if there is a production like A-> "αxβyγ".
Example: S-> iCtSe then i =◦t and t =◦e

x<◦ y – "x has yields precedence to y" if there is a production like A->"αxAB" i.e. x immediately followed by a non-terminal. This non-terminal derives a string A-> γyS , where γ is either ε or single non-terminal, i.e. y is the first terminal symbol derived by A.
Example: S->iC+S then C->b and i <◦ b

x ◦>y - " x has takes precedence over y" if there is a production like A->"αAyB". A-> γxS, i.e. x is the last terminal symbol derived from A.
Example: S->iC+S then C->b and b ◦> +

Precedence Relation Table Construction

It is observed that this parser depends upon a parsing schedule that it must recommend while analyzing a given input string. For example:
E->E+T | T
T->T*F | F
F->id
Firstly find the first terminal and last terminal of each non-terminal.

Table-2
First terminal and last terminal of each non-terminal

Non-Terminal	First Terminal	Last Terminal
E	+,*,id	+,*,id
T	*,id	*,id
F	Id	id

Now find the precedence relation between terminals:

Same Precedence: There is no same precedence between any terminals in this grammar.

Yields precedence:
E->E+T then
+ <◦ *
+ <◦ id

T->T*F then
*<◦ id

$ <◦ +
$ <◦ *
$ <◦ id

Takes precedence:
E-> E+T then
+ ◦> +
*◦> +
id ◦> +

T->T*F then
*◦> *
id ◦> *

+ ◦> $
*◦> $
id ◦> $
For this grammar the precedence relation table is drawn below:

Table-3
Precedence Relation [10]

	id	+	*	$
id	-	◦>	◦>	◦>
+	<◦	◦>	<◦	◦>
*	<◦	◦>	◦>	◦>
$	<◦	<◦	<◦	-

The space of table is n^2 where n is the no. of terminal characters in the grammar.

Implemented Work

Operator precedence parser utilizes precedence functions that represent terminal characters to whole numbers and so the precedence relations between the terminal characters are employed over integral analysis. So for this we design an algorithm by which we can derive precedence functions between terminals.

Algorithm: i. Create functions f_x for each grammar terminal x and for the end of string symbol ($). ii. Partition the symbols in groups so that f_x and g_y are in the same group if x =◦ y. (There can be symbols in the same group even if they are not connected by this relation). iii. Create an inclined sketch whose nodes are in the groups, next for each symbols x and y do: place an edge from the group of g_y to the group of f_x if x<◦ y otherwise if x ◦> y place an edge from the group of f_x to that of g_y. iv. If the constructed sketch has a cycle then no precedence functions exist. When there are no cycles collect the length of the longest paths from the groups of f_x and g_y respectively.

So by this algorithm construct an inclined sketch by which the table II is encoded by two precedence functions f and g that represent terminal characters to whole numbers. Two functions f and g where the following apply:
if x<◦ y then f(x) < g(y)
if x◦> y then f(x) > g(y)
if x=◦y then f(x)= g(y)

To frame these functions construct an inclined sketch with vertices f_i and g_i where i is the i^{th} terminal adapting the given principles:
If x=◦ y then $f_x$and f_y are grouped together and g_x and g_y are grouped together.
If x◦> y then an inclined edge is drawn from f_x to g_y.
If x<◦ y then an inclined edge is drawn to f_x from g_y.

Applying these principles the precedence sketch for the above precedence table looks something like this:

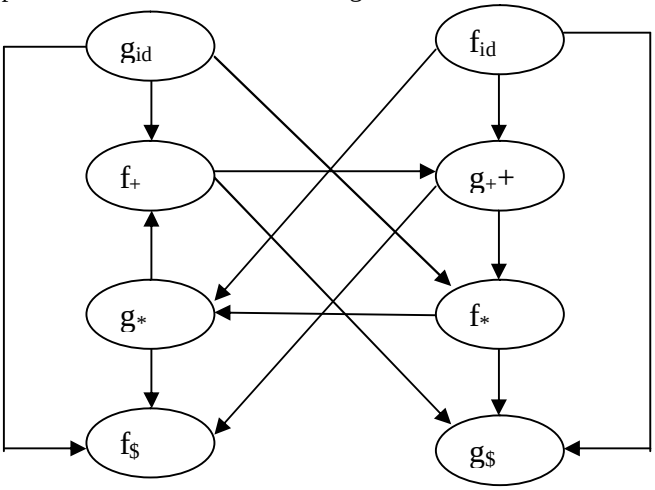

Figure-1
Precedence Sketch

Now it is easy to make a table for the f and g values of each terminal character by signifying f(x) as the long-drawn-out probable route in the sketch starting from f_x such that individual vertex on this route has lower precedence in compare to its nearby preceding vertex, similarly for g(x) as well. The resulting precedence function or long-drawn-out route table driven from above sketch is:

Table-4
Precedence Functions

	id	**+**	*****	**$**
f	4	2	4	0
g	5	1	3	0

Conclusion

Compilers that use operator precedence parser do not need to store the table of precedence relations. The data processing capacity however can be comfortably handled if we construct an efficient data structure to show acceptably the precedence relationship between the terminals. Now it is clear that the data processing capacity is only n*2 where n is the no. of terminal characters. Thus this construction of algorithm for an inclined sketch with the vertices showing functional entities permits preserving a lot of memory. Using this table as a source, the operator precedence parser can parse any string given to it as input. So in this paper our approach provides a fully declarative solution to operator precedence specification for context free grammar. But this parser finds difficulty to hold token that has more than one precedence. So in future our aim is to develop a new method that solves this problem efficiently.

References

1. Peter Ruzicka (1981). Operator Precedence Parsing Algorithm is Polynomial in Time. *Kybernetika,* 17(5), 368-379.

2. Xiao Yang and Jiancheng Wan (2005). A Parsing Algorithm of Natural Language based on Operator Precedence. IEEE: Natural Language Processing and Knowledge Engineering, 73–78, ISBN: 0-7803-9361-9, DOI:10.1109/NLPKE.2005.1598710.

3. Xiao Yang, Jiancheng Wan and Yongbo Qiao (2006). A Binary Combinational Grammar for Chinese and Its Parsing Algorithm. IEEE, 761-766, ISBN: 0-7695-2528-8, DOI: 10.1109/ISDA.2006.253708.

4. Data Syntax and Semantics (2015). http://www-compsci.swan. ac.uk/~csjvt/JVTPublications/DSS (March2006).pdf, 20/07/2015.

5. Alfered V. Aho, Monica S. Lam, Ravi Sethi and Jeffrey D. Ullman (2007). Compilers. Pearson Education Inc., South Asia, 197-200, ISBN-978-81-317-2101-8.

6. WIKIPEDIA (2015) Operator precedence parser, https://en.wikipedia.org/ wiki/Operator-precedence_parser, 15/07/2015.

7. D.S. Henderson and M.R. Levy (1976). An Extended Operator Precedence Parsing Algorithm. *The Computer Journal,* 19(3), 229-233.

8. Arun Petrick (2015). Operator precedence parser, http://compilerdesigndetails.blogspot.in/2012/02/operator-precedence-parsing.html,05/08/2015.

9. Alessandro Barenghi et. al. (2013). PAPAGENO: A Parallel Parser generator for Operator Precedence Grammars. *International Journal of Grid and Utility Computing (IJGUC),* 113(7), 245-249.

10. Shashank Rajput (2015). Operator precedence parser, http://cse.iitkgp.ac.in/~bivasm/notes/scribe/11CS10042.pdf,05/09/2015.