A Conflict in Bottom Up Parsing Approach

Shweta Vasant Shinde
Assistant Professor
Department of Computer Engineering
Vadodara Institute of Engineering, Kotambi, Vadodara, India

Abstract— In the study of compilers there are two approaches used in parsing namely top down and bottom up which are again sub divided. Both the approaches do have limitations. Bottom up parsing includes LR parsing in which there are two types of conflicts reduce – reduce and shift – reduce. The later one can be removed in LALR parsing. The conflict of reduce –reduce is still not solved. The automatic generators use first production rule for grammar causing reduce – reduce conflict but that does not work for all grammars. The basic idea to remove this conflict for all types of grammar is to see the last terminal symbol and first terminal symbol of string and according to it use production.

Key words: Parsing Techniques, Bottom Up Approach Conflict, Look Ahead

I. INTRODUCTION

Parsing is a technique normally which scans the source code and method used to determine how a string of terminals can be produced by a grammar which generally made of production rules. Basically two steps are carried out:

a. Parse tree is generated.

b. Parsers make a single left-to-right scan over the input tokens, look ahead of one terminal at a time, and construct the parse tree. Parsing methods are of two types:

1) Top down parser
2) Bottom Up parser

The first one is also known as top down approach and later bottom up approach respectively.

II. PARSING TECHNIQUES

A. Top-Down Parsing:

The types of parsing in this approach are LL (1) and recursive descent parsing. The parsing steps followed in this approach are:

1) Starting with the root of the parse tree, grow toward leaves.
2) Take a production and match with input string to be parsed.
3) If the production chose is not proper it may need to backtrack.
4) Some grammars don't need backtracking such types of grammars are predictive parsers.

B. Bottom-up Parsers:

The types of parsing techniques in bottom up are LR (1) and operator precedence. The steps followed by the parser are:

1) Start from bottom of parse tree that is leaves and go towards root.
2) As input is parsed, encode possibilities in an internal state.
3) Begin in a state which is valid for legal first tokens.
4) Bottom-up parsers handle a large class of grammars

III. BOTTOM-UP PARSER

In Bottom-up parsing normally we start with the input string sequence and try to apply the production rules in reverse direction, in order to reach or derive the start symbol of grammar. This corresponds to starting at the leaves of the parse tree, and working back to the root. Bottom-up parsing is also known as shift-reduce parsing. LR parsing is a bottom up syntax analysis technique that can be applied to many context free grammars. L is for left –to –right scanning of the input and R for constructing rightmost derivation in reverse.

A. Conflicts faced in SLR Parsers:

1) Shift/Reduce and Reduce/Reduce Conflicts [4]:

1) A state if does not know whether it will make a shift operation or reduction for a terminal, we say that there is a shift/reduce conflict.
2) If a state does not know whether it will make a reduction operation using the production rule i or j for a terminal, we say that there is a reduce/reduce conflict.
3) If the SLR parsing table of a grammar G has a conflict, we say that that grammar is not SLR.

Example: [1]

S -> L=R
S -> R
L -> *R
L -> id
R -> .L
I0: S' -> .S
S -> .L=R
S -> .R
L ->.*R
L -> .id
R ->.L
I1: S'-> S.
I2: S -> L. = R (causes shift/reduce conflict)
R -> .L
I3: S -> R.
I4: L -> *.R
R -> .L
L ->.*R
L -> .id
I5: L -> id
I6: S -> L=.R
R -> .L
L ->.*R
L->.id
I7: L -> *R.
I8: R -> L.
I9: S -> L=R

In these conflicts shift/reduce is removed by LALR Parsers but reduce/reduce problem cannot be completely solved manually.

2) Reduce/Reduce Conflict Resolution:

An approach to remove this conflict by seeing the first or last symbol of the i/p string ,rather than seeing first production rule like in Lark[6] and HYACC[5] ,which to be parsed and see which production rule (only those production rule which causes conflict) can generate these symbols. Our approach require less time in comparison to other approaches because in other approaches we use first production rule for removing conflict and if not getting success to parse the whole i/p string we use another production rule (production rule are those which causes reduce/reduce conflict) while in our approach we only need to see the last or first terminal symbol of input string and according to which we use production rule.

REFERENCES

[1] Bergmann S. "Compiler Design: Theory, Tools, and Examples",4th edition,199, WCB Publishers.
[2] Fraser C., Hanson D., " A Retargetable C Compiler: Design and Implementation",2nd edition,1995 Addison-Wesley Publishing Company.
[3] Aho A, Ullman J. , "The Theory of Parsing, Translation, and Compiling.",3rd edition,2009, Prentice-Hall.
[4] A.A Puntambekar, "Compiler Construction",2nd edition,2009 Technical Publication.
[5] http://hyacc.sourceforge.net, January 27, 2011.
[6] Dr. Josef Grosch. "Cocktail A Tool Box for Compiler Construction Lark - An LALR(2) Parser Generator With Backtracking ",CoCoLab – Datenverarbeitung Achern Germany,Document no. 32,2005.

