

NATURAL LANGUAGE PARSING USING FUZZY CANONICAL LR (1) PARSER

Suvarna G Kanakaraddi¹ and Suvarna S Nandyal²

Abstract- The LR syntax analysis method is a useful and versatile technique for parsing deterministic context-free languages in compiling applications. This paper provides an efficient Fuzzy Canonical LR(1)(FCLR) parser for Fuzzy context-free grammars. The FCLR (1) parser splits states based on differing lookahead sets, it can have many more states than the corresponding Simple LR (1) (SLR) or LR (0) parser. Potentially it could require splitting a state with just one item into a different state for each subset of the possible lookaheads; in a pathological case, this means the entire power set of its follow set. It never actually gets that bad in practice, but a FCLR (1) parser for a programming language might have an order of magnitude more states than an SLR (1) parser.

Keywords- Grammars, Parsers, Compilers, Context-free languages, LR grammars.

I. INTRODUCTION

A compiler for a programming language must verify that its input obeys the syntactic conventions of the language specification. It must also translate its input into an object language program in a manner that is consistent with the semantic specification of the language. In addition, if the input contains syntactic errors, the compiler should announce their presence and try to pinpoint their location. To help perform these functions every compiler has a device within it called a parser [1].

A context-free grammar can be used to help specify the syntax of a programming language. In addition, if the grammar is designed carefully, much of the semantics of the language can be related to the rules of the grammar [2]. There are many different types of parsers for context-free grammars [3]. In this paper we shall restrict ourselves to a class of parsers known as LR parsers. Perhaps more important is the fact that we can automatically generate LR parsers for a large and useful class of context-free grammars. The purpose of this article is to show how LR parsers can be generated from certain context-free grammars. An important feature of the parser generation algorithm is the automatic detection of ambiguities and difficult-to-parse constructs in the language specification. In this paper author discusses LR parsing and outline the parser generation algorithm.

A grammar is used to define a language and to impose a structure on each sentence in the language. We shall be exclusively concerned with Fuzzy context-free grammars, sometimes called BNF (for Backus-Naur form) specifications. In a Fuzzy context-free grammar, we specify two disjoint sets of symbols to help define a language. One is a set of non terminal symbols. We shall represent a non terminal symbol by a string of one or more capital roman letters. For example, LIST represents a non terminal as does the letter A. In the grammar, one non terminal is distinguished as a start (or sentence) symbol. The second set of symbols used in a Fuzzy context-free grammar is the set of terminal symbols. The sentences of the language generated by a grammar will contain only terminal symbols. We shall refer to a terminal or non terminal symbol as a grammar symbol with the Fuzzy membership value attached to it[4]. Next section discusses about the methodology used in the research.

II. METHODOLOGY

A Fuzzy context-free grammar itself consists of a finite set of rules called productions. A production has the form left-side \rightarrow right-side, where left-side is a single non terminal symbol (sometimes called a syntactic category) and right-side is a string of zero or more grammar symbols. The arrow is simply a special symbol that separates the left and right sides. Fuzzy Context-Free Grammar (FCFG), as a straightforward extension of

¹ *Department of Computer Science & Engineering BVBCET Hubli, Karnataka, India*

² *Department of Computer Science & Engineering PDA College of Engineering ,Kalburgi, Karnataka, India*

context-free grammar [4], has been introduced to express uncertainty, ambiguity, and vagueness in natural language fragments [1]. Here production rules are written by considering noun phrase(NP), verb phrase(VP), preposition(PP), adjective (ADJP) [2] and fuzzy membership values are assigned for each rule. Membership values are assigned randomly to each of these rules, finally the values for the entire set of rules for each phrase sums up to 1.

Consider commonly used Production rules for construction of English language sentences are as follows

- | | |
|------------------------------------|-------------------------------------|
| 1) $S \rightarrow NP VP$ (1.0) | 11) $VP \rightarrow v NP$ (0.1) |
| 2) $S \rightarrow aux NP VP$ (1.0) | 12) $VP \rightarrow v VP$ (0.1) |
| 3) $NP \rightarrow art n$ (0.2) | 13) $VP \rightarrow v NP VP$ (0.2) |
| 4) $NP \rightarrow pron$ (0.2) | 14) $VP \rightarrow v ADJP$ (0.1) |
| 5) $NP \rightarrow n$ (0.1) | 15) $VP \rightarrow TO VP$ (0.2) |
| 6) $NP \rightarrow NP PP$ (0.2) | 16) $VP \rightarrow v NP PP$ (0.1) |
| 7) $NP \rightarrow propn$ (0.1) | 17) $VP \rightarrow v PP$ (0.1) |
| 8) $NP \rightarrow NOM$ (0.2) | 18) $PP \rightarrow prep NP$ (1.0) |
| 9) $NOM \rightarrow adj n$ (1.0) | 19) $ADJP \rightarrow adj$ (0.5) |
| 10) $VP \rightarrow v$ (0.1) | 20) $ADJP \rightarrow adj VP$ (0.5) |
| | 21) $TO \rightarrow to$ (1.0) |

A. Parsing

Parser for a grammar to be a device which, when presented with an input string, attempts to construct a derivation tree whose frontier matches the input. If the parser can construct such a derivation tree, then it will have verified that the input string is a sentence of the language generated by the grammar. If the input is syntactically incorrect, then the tree construction process will not succeed and the positions at which the process falters can be used to indicate possible error locations. A parser can operate in many different ways. In this paper we shall restrict ourselves to parsers that examine the input string from left to right, one symbol at a time. These parsers will attempt to construct the derivation tree "bottom-up"; i.e., from the leaves to the root. For historical reasons, these parsers are called LR parsers. The "L" stands for "left-to-right scan of the input", the "R" stands for "rightmost derivation." We shall see that an LR parser operates by reconstructing the reverse of a rightmost derivation for the input. In this section we shall describe in an informal way how a certain class of LR parsers, called LR(1) parse- operate. An LR parser deals with a sequence of partially built trees during its tree construction process. We shall loosely call this sequence of trees a forest. In our framework the forest is built from left to right as the input is read. At a particular stage in the construction process, we have read a certain amount of the input, and we have a partially constructed derivation tree [3]. For example, suppose that we are parsing the input string 'a,b' according to the grammar (ie. After reading the first 'a' we construct the tree.

1) Algorithm for Item Sets:

Set Of Items clousure (I)

```

{
  J = I;
  repeat
    for (each item  $A \xrightarrow{\lambda_i} \alpha . B \beta$  in J) where  $\lambda_i = [0..1]$ 
      for (each production  $B \rightarrow \gamma$  of G)
        if (  $B \xrightarrow{\lambda_i} \gamma$  is not in J )
          add  $B \xrightarrow{\lambda_i} \gamma$  to J ;
  Until no more items are added to J on one round :
  Return J ;
}

```

2) *Fuzzy Shift Reduce Parser Algorithm:*

INPUT: An input string w and Fuzzy parsing tables with functions ACTION and GOTO for a fuzzy context-free grammar.

OUTPUT: If w is in $L(G)$, the reduction steps of bottom up parse for w with its maximum membership value otherwise number of words parsed with its minimum value.

METHOD: Initially the parser has s_0 on its stack, where s_0 is the initial state and $w\$$ is the input buffer.

Let a be the first symbol of $w\$$;

While (1) { /* repeat forever*/

Let s be the state on top of the stack;

If (ACTION [s,a] = shift t) {

Push t on to the stack with respective membership value.

Let a be the next input symbol:

} else if (ACTION [s,a] = reduce $A \rightarrow \beta$) {

Pop $|\beta|$ symbols off the stack:

Let state t now be on top of the stack:

Push GOTO [t,A] on to the stack

Push R_i on to the stack with respective membership value.

Output the production $A \rightarrow \beta$;

} else if (ACTION [s,a] = accept) break;

print result with maximum membership value.

Else call error recovery routine with number of words parsed along with their minimum membership value.

3) *Algorithm for construction of Fuzzy Canonical LR (FCLR) Parsing table:*

INPUT: An augmented grammar G' .

OUTPUT: The Canonical LR Parsing table functions ACTION and GOTO for G' .

METHOD:

1. Construct $C = (I_0, I_1, \dots, I_n)$ the collection of sets of LR(1) Items for G' .

2. State i is constructed from I_i the parsing actions for state i is determined as follows,

a) If $[A \rightarrow \alpha . a \beta]$ is in I_i and $GOTO(I_i, a) = I_j$, then set ACTION [i,a] to "shift j /membership value". Here a must be a terminal.

b) If $[A \rightarrow \alpha .]$ is in I_i , then set ACTION [i, a] to "reduce $A \rightarrow \alpha$ " for all a in FOLLOW(A); here A may not be S' .

c) If $[S' \rightarrow S.]$ is in I_i , then set ACTION [$i, \$$] to "accept".

If any conflicting actions result from the above rules, we say the grammar is not LR (1). The algorithm fails to produce a parser in this case.

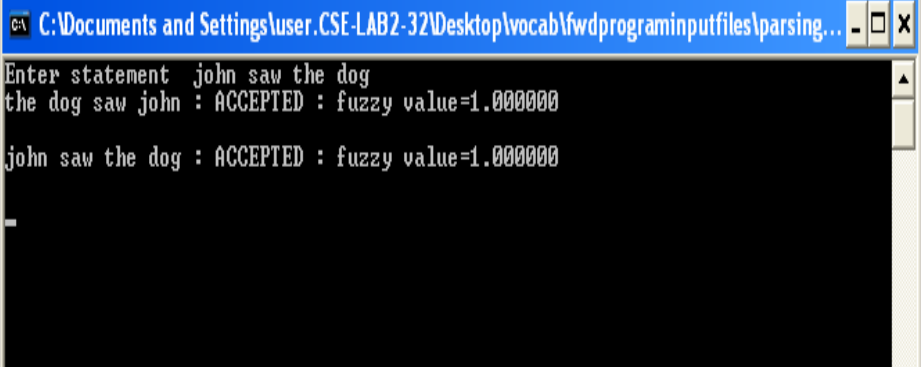
3. The goto transitions for state i are constructed for all non terminals A using the rule : If $GOTO (I_i, A) = I_j$, then $GOTO [i, A] = j$.

4. All entries not defined by rules 2 and 3 are made error.

The initial state of the parser is the one constructed from the set of items containing $[S' \quad .S, \$] \xrightarrow{\lambda_i}$

III RESULTS

Here English sentence input is parsed using Fuzzy Canonical LR parser (FCLR). In this approach initially the grammar is defined and action and goto table is computed, using this table parsing is done for the input sentence. Result shown in Figure 1 depicts the input sentence and result of completely parsed sentence with associated fuzzy value.



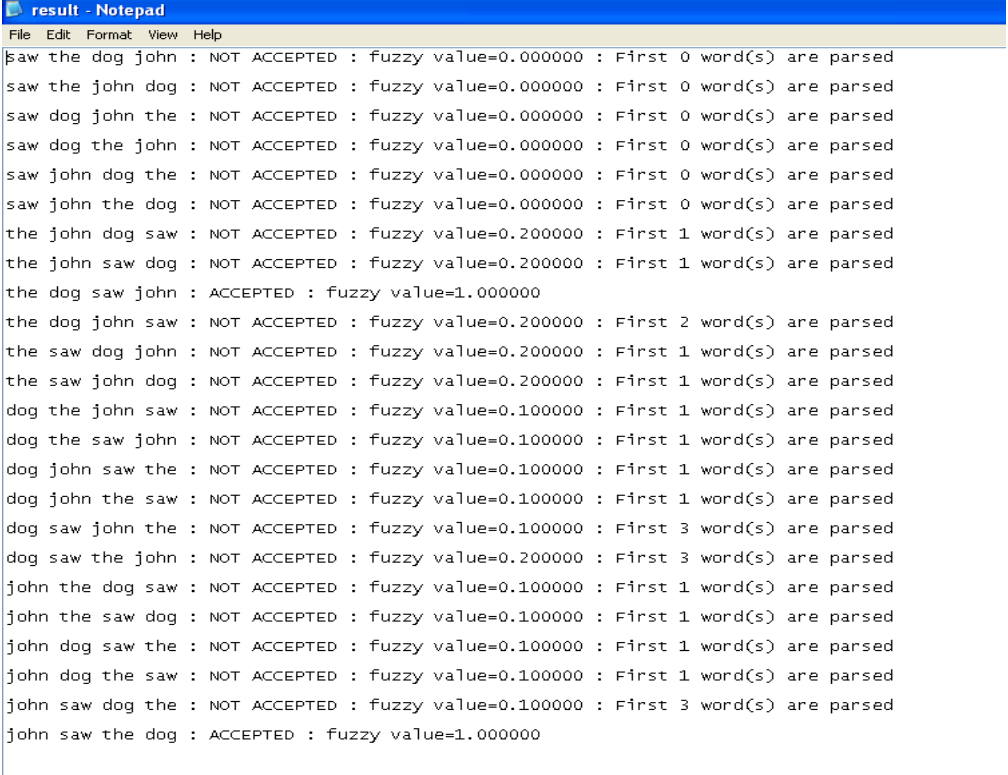
```

C:\Documents and Settings\user.CSE-LAB2-32\Desktop\vocab\fdprograminputfiles\parsing...
Enter statement john saw the dog
the dog saw john : ACCEPTED : fuzzy value=1.000000
john saw the dog : ACCEPTED : fuzzy value=1.000000

```

Figure 1. Parsing result

Following Figure 2 shows the permutations generated and parsing status showing the degree of fuzziness for the input sentence and also shows the completely parsed sentence. Input sentence contains four words so the number of permutations generated for the given input is twenty four permutations. Figure 2 shows the number of words parsed as well as completely parsed sentence with maximum fuzzy membership value as 1, which shows that syntactically correct sentences will get the maximum fuzzy membership value as 1.



```

result - Notepad
File Edit Format View Help
saw the dog john : NOT ACCEPTED : fuzzy value=0.000000 : First 0 word(s) are parsed
saw the john dog : NOT ACCEPTED : fuzzy value=0.000000 : First 0 word(s) are parsed
saw dog john the : NOT ACCEPTED : fuzzy value=0.000000 : First 0 word(s) are parsed
saw dog the john : NOT ACCEPTED : fuzzy value=0.000000 : First 0 word(s) are parsed
saw john dog the : NOT ACCEPTED : fuzzy value=0.000000 : First 0 word(s) are parsed
saw john the dog : NOT ACCEPTED : fuzzy value=0.000000 : First 0 word(s) are parsed
the john dog saw : NOT ACCEPTED : fuzzy value=0.200000 : First 1 word(s) are parsed
the john saw dog : NOT ACCEPTED : fuzzy value=0.200000 : First 1 word(s) are parsed
the dog saw john : ACCEPTED : fuzzy value=1.000000
the dog john saw : NOT ACCEPTED : fuzzy value=0.200000 : First 2 word(s) are parsed
the saw dog john : NOT ACCEPTED : fuzzy value=0.200000 : First 1 word(s) are parsed
the saw john dog : NOT ACCEPTED : fuzzy value=0.200000 : First 1 word(s) are parsed
dog the john saw : NOT ACCEPTED : fuzzy value=0.100000 : First 1 word(s) are parsed
dog the saw john : NOT ACCEPTED : fuzzy value=0.100000 : First 1 word(s) are parsed
dog john saw the : NOT ACCEPTED : fuzzy value=0.100000 : First 1 word(s) are parsed
dog john the saw : NOT ACCEPTED : fuzzy value=0.100000 : First 1 word(s) are parsed
dog saw john the : NOT ACCEPTED : fuzzy value=0.100000 : First 3 word(s) are parsed
dog saw the john : NOT ACCEPTED : fuzzy value=0.200000 : First 3 word(s) are parsed
john the dog saw : NOT ACCEPTED : fuzzy value=0.100000 : First 1 word(s) are parsed
john the saw dog : NOT ACCEPTED : fuzzy value=0.100000 : First 1 word(s) are parsed
john dog saw the : NOT ACCEPTED : fuzzy value=0.100000 : First 1 word(s) are parsed
john dog the saw : NOT ACCEPTED : fuzzy value=0.100000 : First 1 word(s) are parsed
john saw dog the : NOT ACCEPTED : fuzzy value=0.100000 : First 3 word(s) are parsed
john saw the dog : ACCEPTED : fuzzy value=1.000000

```

Figure 2. Permutations of FCLR input

IV. CONCLUSION

In this paper author has developed a novel parsing technique called Fuzzy Canonical LR (FCLR), Here Fuzzy context free grammar is designed for parsing Natural language. Authors have implemented FCLR algorithm in 'C' Programming Language. Considering English language sentence as an input, permutations are generated and for the generated permutations Fuzzy Canonical LR algorithm are applied. Finally Fuzzy max-min technique is applied to get the degree of fuzziness. Experimental results have been presented here. Our research work involves the design of fuzzy parsing algorithms and implementation to provide the better results compare to conventional approach. The main advantage of fuzzy parsers over conventional parser is that it gives degree of fuzziness and syntactic correctness for partially parsed sentences but in conventional parsers the sentences parsed completely are only accepted and rejected completely if it is partially parsed. Syntax analysis helps to improve recognition rates significantly. Authors conclude that one of the major drawbacks of FCLR algorithm is number of states generated are more compare to Fuzzy simple LR (FSLR) algorithm.

REFERENCES

- [1]. Hejab M. Alfawareh, Shaidah Jusoh, "Resolving Ambiguous Entity through Context Knowledge and Fuzzy Approach", International Journal on Computer Science and Engineering (IJCSE), Vol. 3 No. 1 Jan 2011.
- [2]. Erkki Luuk, The noun/verb and predicate/argument structures, Lingua 119, Elsevier Publication , pp 1707–1727, 2009.
- [3]. Alfred V Aho, "Compilers Principles, Techniques, and Tools", Pearson Education, pp.191-217.
- [4]. John N Mordeson, "Fuzzy Automata and Languages", Chapman & Hall/CRC Crc Press company, Washington D.C, pp. 127-137.