Binish *et al.LGU (IJECI)* 2020

LGU (IJECI)
ISSN: 2522-3429 (Print)
ISSN: 2616-6003 (Online)

LGU International Journal for
Electronic Crime Investigation

Research Article

Vol. 4 issue 4 Year 2020

LALR Parser Implementation using Grammar Rules

Syeda Binish Zahra
binishzahra@gmail.com
National College of Business
Administration & Economics (NCBAE

Abstract:
Syntactic parsing deals with syntactic structure of a sentence. It refers to the grammatical arrangement of words in a complete sentence. A syntactic analysis of English words will be presented using bottom up parsing in which LALR parser defines the best syntax analysis. A compiler is constructed that generates token of the identifiers that are characters, alphabets, etc. using symbol table, checks for the syntax of sentence by dividing into phrases, generating errors, debugging them and giving us the correct sentence.

Keyword:

1. Introduction

Compiler as we know is a program that takes one language as an input and translates into another equipment language which we plea. Compilers are broadly applicable and are used frequently in many unexpected areas. A compiler is said to be good that contain self-contained units that are ready to be executed [1]. Analysis and synthesis are two parts of compiler that further divides the operation of compiler in six phases.

Grammar is a set of language rules to create phases and sentences that convey meaning and those rules which involve meaning of words are called semantics and those that don't involve meaning are called syntactic. So both the terminologies mean a lot. Context free grammar (CFG) are the set of rules or productions that shows which element occur in a phase and in what order.

Many researches have been published on parsing methods for natural language. Here we want to parse the grammar of English language [2, 3]. Simply, there is a need of compiler that translates the syntax of English language. Syntax analysis is a fundamental area of research and is used in key areas of computational etymology for example in machine translation, information retrieval etc. this determination of syntax analysis is done by the help of parsing.

The main purpose of Syntax analyzer is to identify the syntactic structure of a sentence and parsing them accordingly. A widely used mathematical system for exhibiting essential structure in natural language is context-free grammar (CFG) also known as phrase structure grammar. Parse trees are used to show the structure of the sentence, but they often contain dismissed information due to understood definitions [1]. Context-free grammar (CFG) was first defined for natural language by Chomsky in 1957 that consist of terminals, non-terminals and a production rule from non-terminal to terminal or another non-terminal.

Methodology:

Many online and offline service/software are provided to frequently check the errors in our grammar for both syntax and semantic error. Researches are proposed and published on the parsing of natural languages. In the previous research, Parse trees are used to show the structure of sentence [4]. Firstly it shows the type of sentence, then the components of the sentence are identified. Again the grammar rules are checked, if the sentence parse through the defined grammar then the sentence is considered as syntactically correct. Otherwise it is syntactically incorrect. The context free grammar of the symbol will be defined along with the symbol table. The parts of speech tagger will also be the part of the compiler.

In many researches, the LL1 parser are used as top down parsing that starts checking from the start point mainly ROOT and comes downward towards the LEAVES (Ending point). Here the advantage of use of this is that it never waste time on subtrees and can go further deep to find the valid string. Rule based approach algorithms and part of speech tagger are used as the part of software. But the issue is that it waste time on trees that don't match the input. It compares the first word of the input with the leftmost branch of the tree. The need of writing this paper is the use of bottom up parsing such as LALR. As in some previous research papers, top down parsing are used.

Here in this research we will use bottom up parser that starts from the words in the input sentence and attempts to construct a parse tree in an upward direction towards the root. At each step or level the parser with look for the rules and the defined productions. The idea is to use LALR parser because of its effectiveness [5]. LALR stands for look ahead left right is a technique for deciding when reductions have to be made in shift/reduce parsing. Often, it can make the decisions without using a look ahead. Sometimes, a look ahead of 1 is required [6, 7].

Most parser generators construct LALR parsers. In LALR parsing, a deterministic finite automaton is used for determining when reductions have to be made. The deterministic finite automaton is usually called prefix automaton [8]. This look ahead parser uses look ahead sets. If a state has more than one reduction, or a reduction and a shift, the parser looks at the look ahead symbol, in order to decide what to do next. With LALR (look ahead LR) parsing, we attempt to reduce the number of states in an LR (1) by merging similar states. This reduce the number of states to the same as SLR (1) but still holds some powers of the LR (1) look ahead.

LALR parser starts with the idea of building an LR parsing table. These generated tables are less powerful than LR but more than SLR Techniques.

Evaluation:

Bottom up parsing mainly focuses on shift reduce parser in which the stack hold grammar symbol and an input buffer holds the best of string to be parsed. But there are some limitation in shift reduce parser. Sometimes in this parser we need to look ahead. So, knowing the benefit of LALR parser we can easily check the grammar of our language. The context free grammar defines the production rules. Setting the production rules according to the program. The symbol table that adds new identifiers in memory, generate tokens according to their values. Here we are using English language so alphabets, special characters will be added as identifiers. Special and most frequently used words might also be stored for the semantic meaning of the sentence.

Using the LALR parser, the sentence can effortlessly be parsed with look ahead. Here the grammar in the CFG is Verb, pronoun, nouns, articles, adjectives, and other grammatical verses. The CFG just defines syntax but the structures are not specified. Firstly turning a string/file into a series of tokens during a phase referred to as "Lexical Analysis" [3]. Once we have a collection of tokens, we match them against a "grammar." Grammars are simple languages that are used to bootstrap more complex languages [2]. They consist of simple mapping rules that indicate what rule to evaluate next. Then the production rules are checked and proceeds accordingly.

The LALR parser has more language recognition power than the LR (0) parser, while requiring the same number of states as the LR (0) parser for a language that can be recognized by both parsers. This makes the LALR parser a memory-efficient alternative to the LR (1) parser for languages that are not LR (0).

Conclusion and Future Work:

This paper focuses on the syntax analysis for natural language using LALR parser. This is an approach to check the correctness of the sentence. This approach cannot achieve the accuracy and checking up to 100 % but still works accurately. Here in future its performance can also be enhanced and this can further be enhanced by constructing a compiler that translates URDU words using this technique.

References:

[2] Haider, H.; Rosengren, I. Scrambling; Sprache und Pragmatik: Lund, Sweden, 1998.

[3] Kübler, S.; McDonald, R.; Nivre, J. Dependency parsing. Synth. Lect. Hum.Lang. Technol. 2009, 1, 1–127.

[4] Kuhlmann, M. Mildly non-projective dependency grammar. Comput. Linguist. 2013, 39, 355–387.

[5] Skut, W.; Krenn, B.; Brants, T.; Uszkoreit, H. An annotation scheme for free word order languages. In Proceedings of the 5th Applied Natural Language Processing Conference, 31 March–3 April 1997; pp. 88–95.

[6] Ranta, A. Grammatical Framework: Programming with Multilingual Grammars; CSLI Publications: Stanford, CA, USA, 2011.

[7] Ljunglöf, P. Expressivity and Complexity of the Grammatical Framework. Ph.D. Thesis, Göteborg University, Gothenburg, Sweden, 2004.

[8] Kallmeyer, L.; Maier, W.; Parmentier, Y.; Dellert, J. TuLiPA-Parsing extensions of TAG with range concatenation grammars. Bull. Pol. Acad. Sci. 2010, 58, 377–391.

[9] Kallmeyer, L.; Parmentier, Y. On the relation between multicomponent tree adjoining grammars with tree tuples (TT-MCTAG) and range concatenation grammars (RCG). In Proceedings of the Second International Conference on Language and Automata Theory and Applications (LATA 2008), Tarragona, Spain, 13–19 March 2008; pp. 263–274.