Scientific Journal of Impact Factor (SJIF): 4.72

e-ISSN (O): 2348-4470
p-ISSN (P): 2348-6406

International Journal of Advance Engineering and Research Development

Volume 4, Issue 4, April -2017

Implementation of User Interface for LL(k) Parser Generator

Prof S.R Dhore[1], Alok Singh[2], Irale Yogesh Lahu[3], Pankaj Kumar[4], Rohit Rawat[5]

[1,2,3,4,5] *Department of Computer Engineering, Army Institute of Technology, Pune, India*

Abstract —*A visual parser-generator application for generating parsers without any textual grammar specification, script or code. Unlike other parser-generators, it represents parser rules as visual grammar trees with distinct icons for the grammar-tree nodes. This application provides facilities of generation of abstract syntax tree. Along with user friendly interface this application allows the user to store their grammar as back in an XML file that can be later used for reviewing, testing, or modification of grammar. This application accepts context-free grammars from the user for parser generation. It allows users to develop, edit and understand the working and flow of grammar languages and also facilitate them to test their own grammar that is user can made their own grammar and test on this application. This application is pretty user friendly and provide a handy graphical user interface environment in generating parse trees and action code generation.*

Keywords- Regular Expression, Abstract Syntax Tree,Context Free Grammar, Bottom up parsing, First and Follow, Production rules, Grammar Tree, Parser, LL(K) Grammar.

I. INTRODUCTION

The idea involves with the development of visual parser-generator application for generating parsers without any textual grammar specification, script or code . Here, the parser rules are represented as visual grammar trees with distinct icons for the grammar-tree nodes. The basic idea and motivation behind developing this project is to help out the users to learn about the parsing techniques easily. Through this project we are creating a learning tool which implements the processing of a regular expression using LL(k) parser generator and help users in quickly understand grammar-trees.

For this we are providing numerous menus for editing grammar-trees in parsers The purpose of this project is to provide users a menu driven tool and other editing options or tools. To learn to process text input using both regular expressions and parser generators. To create a learning tool which implements the processing of a regular expression using LL(k) parser generator. The parser generator consists of user defined grammar tree i.e a collection of tokens and expressions with user defined rules,action code, .this will help out the users to learn about the parsing techniques easily.

Users need tools for understanding and solving their problems in efficient and flexible manner. They want to automate their tasks and generate output that can be easily applied into their application. This application provides facilities for automatic abstract syntax tree construction. The development of the project is based according to the basic knowledge of the user and their curiosity of understanding the basic implement of language parser and how the parser works with the specific grammar rules. The software is also designed keeping in mind the general understanding level of user such that they can easily modify the grammar rules and develop their own modified language.

II. RELATED WORK

There are various educational tools available which focuses on providing theoretical as well as practical information on the topic of parsing. Some tools focuses on theoretical aspects while some on practical. These tools can handle wide variety of parsing algorithms like LL(1), LL(k), SLR(1) and LALR(1).

Various attempts have been made to fill the difference between theoretical and practical domain in parser generation field. One vary famous tool available is JFLAP. It is more theory oriented tool, it helped student in the generation of the head and follow sets or the processing of a given input stream. This JFLAP tool works very well with the theory part but failed when it comes to practical aspect and does not allow students to generate their own parsers.

In the early stages of tools generation, they were limited in terms of parsing user specified grammars. They only works with LL(1). Later on some tools were developed which allowed users to parse their own specified grammar. One such tool is VCOCO. It allows users to parse their own specified grammar. But talking about its drawback the tool is mainly designed for experts rather than for educational purpose for students.

Later on with the gain in knowledge in the fields of parsing algorithms some more advanced tools have been developed. ANTLR is one such tool, it is first of its kind. It accepts left-recursive grammars from the user which can be recognized as context-sensitive language. It provides support for debugging and error handling. It generate DFA which can be used for Back-Tracking when it's operation gets fail. But, this tool lacks in providing user interface to the users, this might be its limitation, rest this tool is very good for developers as well as for students.

Many efforts later on have been put in providing interface to the users for better understanding of parser languages. One such tool is VAST(Visualization of Abstract Syntax Tree). Its main focus is on providing user interface to the learners. It represents visualization of Abstract Syntax Tree completely independent of the Parser Generator. This tool is not good in dealing with large trees. All above tools either works with theory part or with practical part and do not provide any information about how parsing rules look like or the code part representing the parsing rule.

III. VISUALIZATION

The application is mainly divided into five modules- Parser rules, action code, abstract syntax tree, Input- Regular expressions , Output- Log. Parser rules are represented in the hierarchical tree structure with each nodes associated with some symbol and optional action code. Action code is the function taking two or more arguments as input for the required node of the grammar tree and implementing it whenever a regular expression is parsed.

4.1. Parser Test Input

Regular Expressions are used as Input to Parser which is stored in an XML file. Parser test input is involved in order to test the grammar rule created by the user. Testing reflects that how the grammar rules and abstract syntax tree behaves with different type of regular expressions.

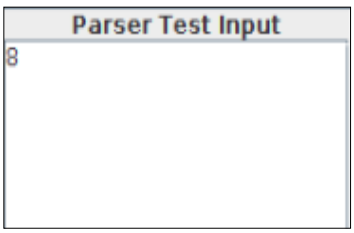


Figure 1: Example of Parser Text Input

4.2. Grammar Tree

The Grammar Tree will be at the top left of GUI. It represents the structure of selected parser-rule. A pop-up context menu is displayed when any node of the grammar-tree right clicked. This menu will also supports grammar-tree editing functions.

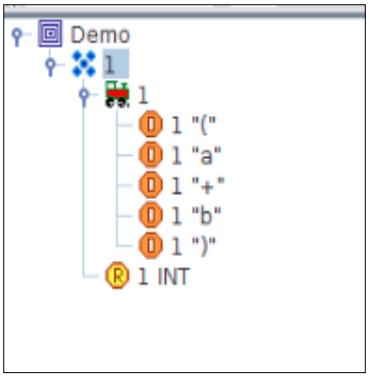


Figure 2: Example of Grammar Tree

4.3. Parse Tree Structure

Abstract syntax tree is a data structure that will be created according to the grammar rules specified. Input consists of a regular expression with combination of operators and operands. Output consist of a log generated for the inputed regular expression.

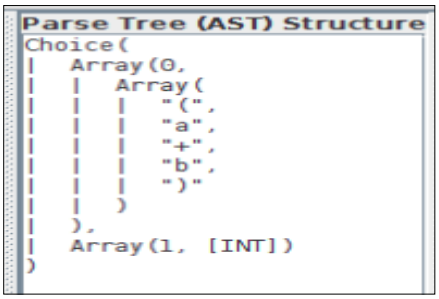


Figure 3: Example of Parse Tree

4.4. Action Code

A function literal, coded in Java script or Scala, associated with any grammar-tree node (also known as semantic action or just action in CS theory). The function is executed when AST matches parser input to the node. This part is optional to the user.

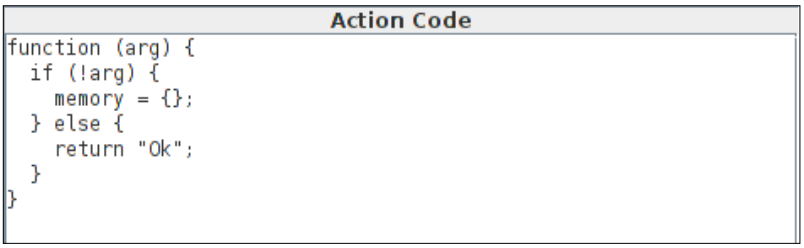


Figure 4: Example of Action Code

4.5. Parser Log

The generated result of the testing can be seen at the Parser Log area. This section will basically display the output for the input specified. If the corresponding action code is provided to the AST then the output generated will be the solution to the expression provided in input else it will be the representation of the expression in AST manner will be displayed.

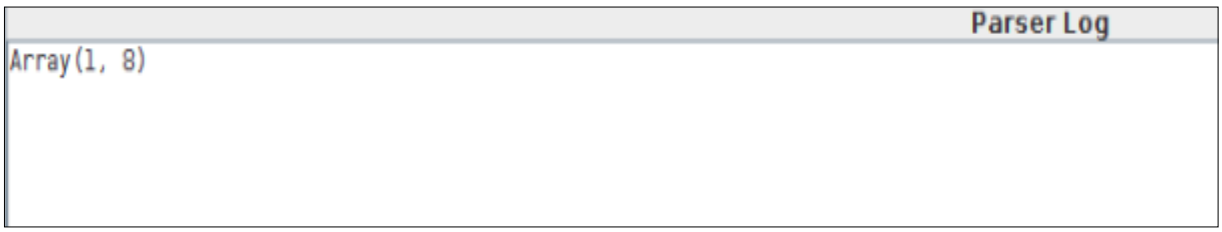


Figure 5: Example of Parser Log

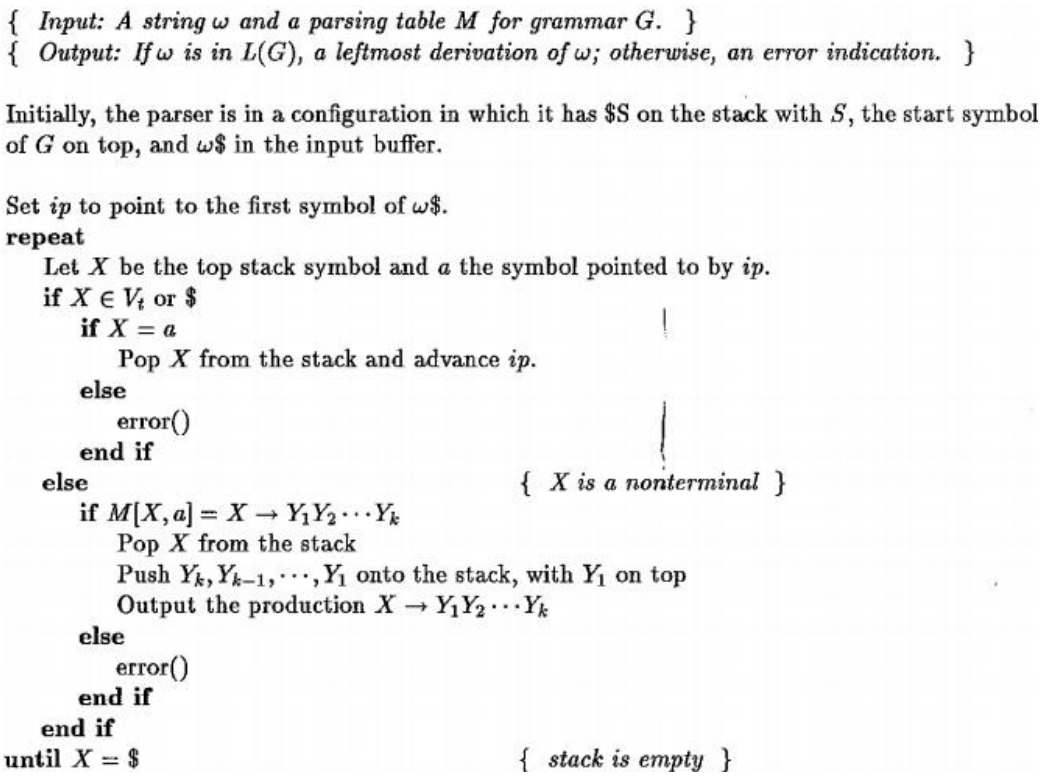
IV. IMPLEMENTATION

The whole Graphical User Interface part of the application is developed using the Java Abstract Window Toolkit (AWT). Combinators API is used by client program to load file which contain grammar and creates parser from it. This API is written in JAVA which can be used by any JVM platform. This API is pretty easy to understand and it requires good understanding of Syntax Tree.

This API contains four important functions i.e. Main(), ExprFun(), TermFun(), FactorFun(). These function used to generate AST according to their functionality or rule which is defined in the block. AST is passed by the main() function to another function so another function need not to use API. They contains JAVA codes. It uses LL(k) algorithm to implement LL(k) parser.

3.1. LL(k) Parsing Algorithm

The LL parser is a deterministic pushdown automation with the ability to peek on the next k input symbols without reading. This capability can be emulated by storing the lookahead buffer contents in the finite state space, since both buffer and input alphabet are finite in size. As a result, this does not make the automaton more powerful, but is a convenient abstraction.



{ *Input: A string ω and a parsing table M for grammar G.* }
{ *Output: If ω is in $L(G)$, a leftmost derivation of ω; otherwise, an error indication.* }

Initially, the parser is in a configuration in which it has \$S on the stack with S, the start symbol of G on top, and ω\$ in the input buffer.

Set *ip* to point to the first symbol of ω\$.
repeat
  Let X be the top stack symbol and a the symbol pointed to by *ip*.
  if $X \in V_t$ or \$
    if $X = a$
      Pop X from the stack and advance *ip*.
    else
      error()
    end if
  else { *X is a nonterminal* }
    if $M[X, a] = X \rightarrow Y_1 Y_2 \cdots Y_k$
      Pop X from the stack
      Push $Y_k, Y_{k-1}, \cdots, Y_1$ onto the stack, with Y_1 on top
      Output the production $X \rightarrow Y_1 Y_2 \cdots Y_k$
    else
      error()
    end if
  end if
until $X =$ \$ { *stack is empty* }

Figure 5: LL(1) parse algorithm

V. APPLICATION

This IDE has various application in use:
1) It allows development of parsers without any code or script of any kind with user friendly interface.
2) Using this application or IDE we can help people with a little bit of parser knowledge to develop their own grammar. User can learn by his own how the rules are parsed and how we can create that rules.
3) It supports automatic parser generation and handles a large number of grammar classes. Wide number of parser generation rules are available for the user.
4) It is designed for two point of view :
  (i) Usability Purpose.
  (ii) Educational points of view.

VI. CONCLUSION AND FUTURE WORK

We have designed a visual parser-generator IDE for developing parsers without any textual grammar specification, script or code representing parser rules as visual grammar-trees with distinct icons for the grammar-tree nodes. They are also executable, and can be run at any time at the click of a button. Users need tools for understanding and solving their problems in efficient and flexible manner. They want to automate their tasks and generate output that can be easily applied into their application.

Some highlighted features of this includes:
(1). It accepts a context-free grammar.
(2). It accepts a regular expression as terminal symbols.
(3). It allows the programmers to interact with the grammar without an explicit code generation step.
(4). It provides an API that enables the programmers to write tree-walking code quickly and easily.

The main focus in further development of application involves more interactiveness and including other parsing techniques for educational purposes. Its main focus will be on making the project more user friendly and easy to use for students as well as for developers. Along with this we will try to improve navigation of regular expressions using different parsing techniques which will help user in understanding other more efficient parsing techniques.

REFERENCES

[1] Incremental LL(1) Parsing in Language-Based Editors John J. Shilling

[2] ANTLR: A Predicate LL(k) Parser Generator T. J. P ARR University of Minnesota, AHPCRC, 1100 Washington Ave S Ste 101, Minneapolis

[3] Visualization of Syntax Trees for Language Processing Courses. Francisco J. Almeida-Mart nez, Jaime Urquiza Fuentes J. Angel Vel azquez-Iturbide

[4] Programming Language Processors in Java David A. Watt and Deryck F. Brown, Pearson Education

[5] Modern Compiler Implementation in Java A.W. Appel, Cambridge University Press[6] Compilers: Principles, Techniques and Tools A.V. Aho, R.Sethi, and J.D. Ullman, Addison Wesley

