International Journal of Innovative Research in Advanced Engineering (IJIRAE)
Volume 1 Issue1 (March 2014)

A TOOL FOR VISUALISATION OF PARSERS: JFLAP

S.Devakumar
Department of CSE
Vignan's University,INDIA
deva.248@gmail.com

D.S.Bhupal Naik,
Department of CSE
Vignan's University,INDIA
dsbhupal@gmail.com

S.V.Ramakrishna
Department of CSE
Vignan's University,INDIA
bobbysajja@gmail.com

Abstract— In recent years, tools for computer aided learning have become widespread on all levels of education. These tools are used as alternatives or additions to traditional methods of teaching, so to wastage of student time to absorb the taught course. On the other hand, these tools are very much helpful to the students those who are having the burden of course lecturers when preparing complex examples since these can be created and executed in real time and then analyzed in class. To gain knowledge other than the regular methods, we have developed JFLAP tool, a tool for learning basic concepts of Formal Languages and Automata Theory. The main goal of the JFLAP tool is to create Type 0 languages, Type 1 languages, Type 2 languages, Type 4 languages, Parsers and visually present complex concepts and mathematical models of Automata Theory fundamentals. In this paper, we propose, JFLAP converts user defined i.e. Context Free Grammar to SLR (Simple LR) parsing table and presents it to the user in a form of state diagram, Parsing Table and Derivation Table.

Keywords: JFLAP, Context-Free Grammars, LR Parsers and Computer aided learning, Context-Free language.

I. INTRODUCTION

Parser or Syntax analyzer obtains a string of tokens from lexical analyzer and verifies that it can be generated by the language for the source program. The Syntax analyzer should generate syntactical errors in an intelligible fashion. The two types of parsers employed are 1. Top Down Parser: which construct a syntax tree or parse tree from top (root) to bottom (leaves), examples are Recursive Descent Parser, Predictive Parser, LL(1) Parser. 2. Bottom Up Parser: which construct a syntax tree or parse trees from leaves and work up the root, examples are Shift-Reduce Parser, Operator Precedence, LR Parsers – Simple LR, Canonical LR and Look head LR. In LR parser, 'L' is for left to right scanning of the input and the 'R' is for constructing a rightmost derivation in reverse. Why we are using LR parsers, the reason is it can be constructed to recognize virtually all programming language constructs for which context free grammars can be written. The class of grammars that can be parsed using LR methods is a proper subset of the class of grammars that can be parsed with predictive parsers. It can also detect a syntactic error as soon as it is possible to do so on a left to right scan of the input.

We have developed the JFLAP tool to help both students and lecturers. The tool generates Context Free Grammar of different complexity and converts them into corresponding SLR parsing table [1]. The conversion algorithm used within the tool is explained in every course that deals with automata theory, since it is a proof by construction that every Context Free Grammar can be converted to a SLR parsing table [2]. The tool allows students to experiment with various context free grammars and see how the changes made affect the resulting SLR parsing. Learning through experimentation is proven to be one of the most efficient ways of education, since the best way to understand an algorithm or method is to see it applied on a live example. Furthermore, the tool also enables the lecturer to make the lecture more interesting. The lecturer can also skip the detailed description of every step of the algorithm, since the students can easily analyze the steps from the listing at their convenience. JFLAP tool allows easy construction of different problems of approximately the same difficulty for tests and verification of the students' solutions.

II. LITERATURE SURVEY

Computer aided learning tools are today available for almost any field of science. One of the most popular such tools, used extensively in both education and research, is *Mathematica* [3]. *Mathematica* provides a high-level, interpreted programming language, and offers vast numerical and graphical libraries. The tool also contains *Combinatorica* [4], a collection of over 450 algorithms for discrete mathematics and graph theory. Simpler, non-commercial tools are also available on the Web. For example, detailed animations of the basic sorting algorithms as well as the source code used can be found at [5]. For computer networks, animations of the various data-link layer protocols can be found at [6]. In the field of regular languages and automata theory, several notable tools exist. *FIRE engine* [7] is a C++ class library implementing finite automata and regular expression algorithms. *Grail+* [8] is a symbolic computation environment for finite-state machines, regular expressions, and other formal language theory objects. Using *Grail+*, one can input

machines or expressions, convert them from one form to another, minimize them, complement them, and make them deterministic. *JFLAP* [9] is software for experimenting with formal language topics including Finite Automata, Mealy Machine, Moore Machine, Pushdown Automata, Turing Machine, Multi-Tape Turing Machine, Grammar, L-System, Regular Expression, Regular Pumping Lemma and Context-Free Pumping Lemma. Prior to *RegExpert*, we have developed *Automata Simulator* [10] and *SoftLab* [11], both applicable in the field of automata theory fundamentals. *Automata Simulator* offers interactive generation and simulation of finite state machines. It supports all types of finite state machines: automata with binary output (DFA, NFA, and NFA-epsilon) and automata with general output (Moore and Mealy automata). *Softlab* is an extension of *Automata Simulator* towards a fully distributed e-learning tool. It allows better interaction between the student and the supervisor as the supervisor can monitor the student's progress and even assist in modelling more complex examples.

III. EXISTING METHOD

A Simple LR parser or SLR parser is an LR parser for which the parsing tables are generated as for an LR(0) parser except that it only performs a reduction with a grammar rule $A \rightarrow w$ if the next symbol on the input stream is in the follow set of A*. Such a parser can prevent certain shift-reduce and reduce-reduce conflicts that occur in LR (0) parsers and it can therefore deal with more grammars. Sometimes and some cases the parser cannot parsed by an LR (1) parser. A grammar that can be parsed by an SLR parser is called a *SLR grammar*. The construction of SLR parsing table obtaining in two steps. In step 1, construction of set of items from the augmented grammar and in step 2, find goto graph from the set of items formed.

IV. PROPOSED METHOD

In this paper, we proposed a tool called JFLAP. The tool consists of eleven main components shown in the Fig 1: Finite Automata, Mealy Machine, Moore Machine, Pushdown Automata, Turing Machine, Multi-Tape Turing Machine, Grammar, L-System, Regular Expression, Regular Pumping Lemma and Context-Free Pumping Lemma. In this paper we propose Grammar components, so users input Context-Free grammar by manually, which generates SLR parsing table automatically. The former approach is useful to students for evaluating their paperwork solutions, while the latter approach allows for experimentation with different classes of expressions and helps lecturers to prepare student assignments.

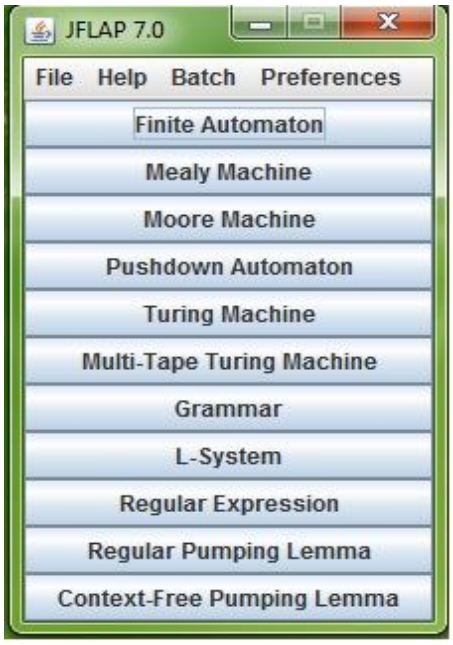


Fig 1

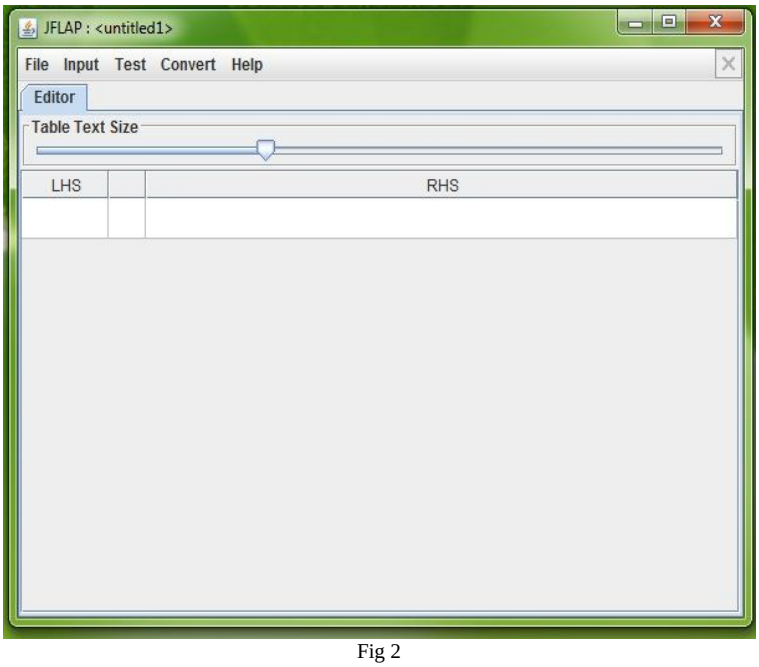
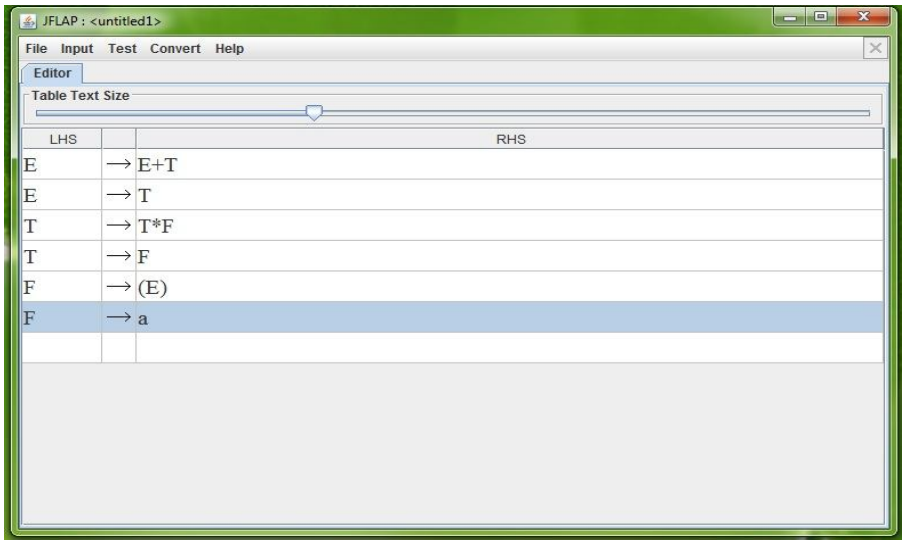


Fig 2

The Fig 1 shows the JFLAP components, we can click on Grammar button then shown in Fig 2 will be displayed then enter that grammar. After entering the grammar shown in Fig 3, click on Input and then Build SLR (1) Parse Table. A

new rule is automatically added to the grammar, “E'->E”, resulting in a new start variable, “E”, that does not appear in any right-hand sides.



Next, we will go to Input tab and select Build SLR (1), first it construct augmented grammar then find FIRST and FOLLOW functions of the augmented grammar. We now build a DFA that models the SLR (1) process. Each state will have a label with marked productions called items. The marker is shown as “.”. Those symbols to the left of the marker in an item are already on the parsing stack, and those symbols to the right of the marker still need to be pushed on the parsing stack. All the items for one state are called an item set. JFLAP now asks you whether you want to define the initial set or not. If you click “Yes”, JFLAP will pop up a small window where you can define the initial set. If you click “No”, then JFLAP will automatically create the initial set for you. After that it creates list of items, based on these items the transition diagram will be created then parsing table should be filled with by using action and goto functions shown in Fig 4. Finally the derivation table should be displayed shown in Fig 5.

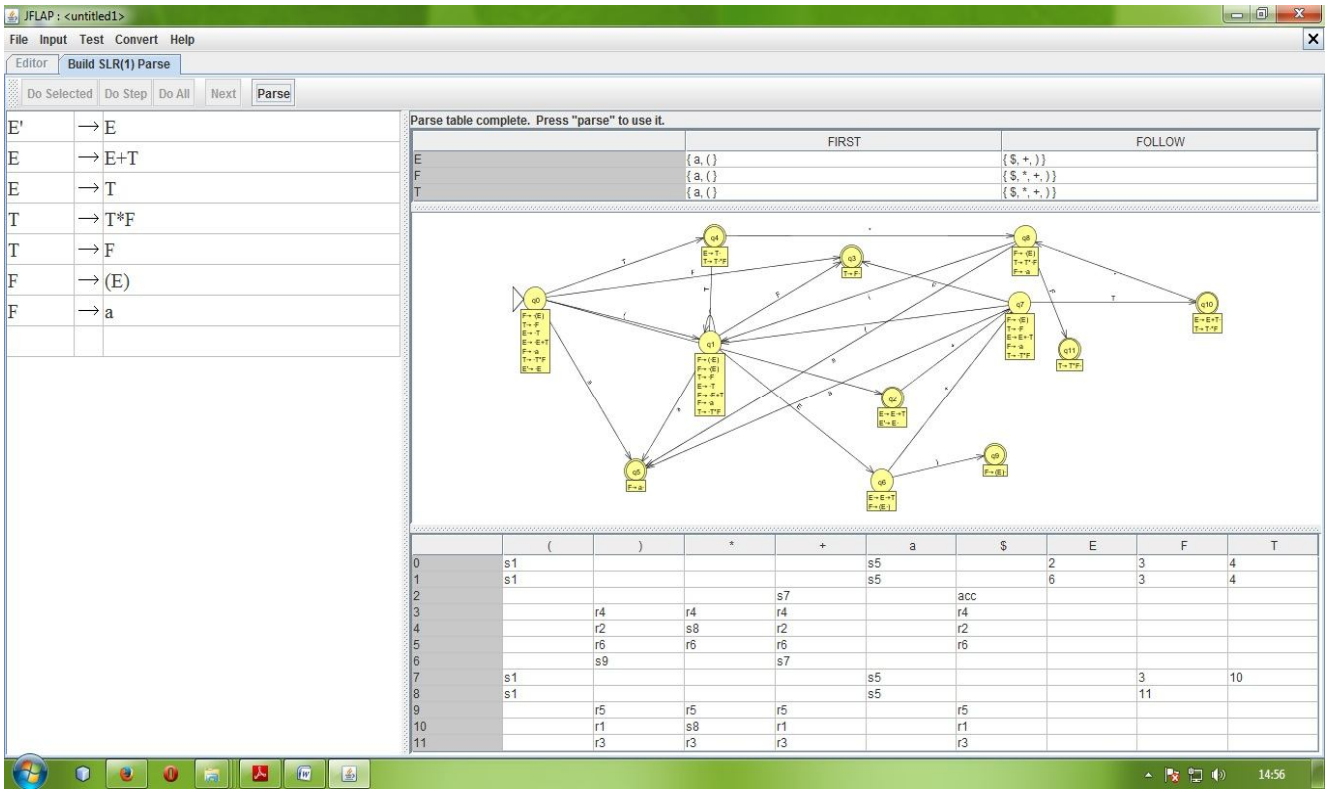


Fig 4

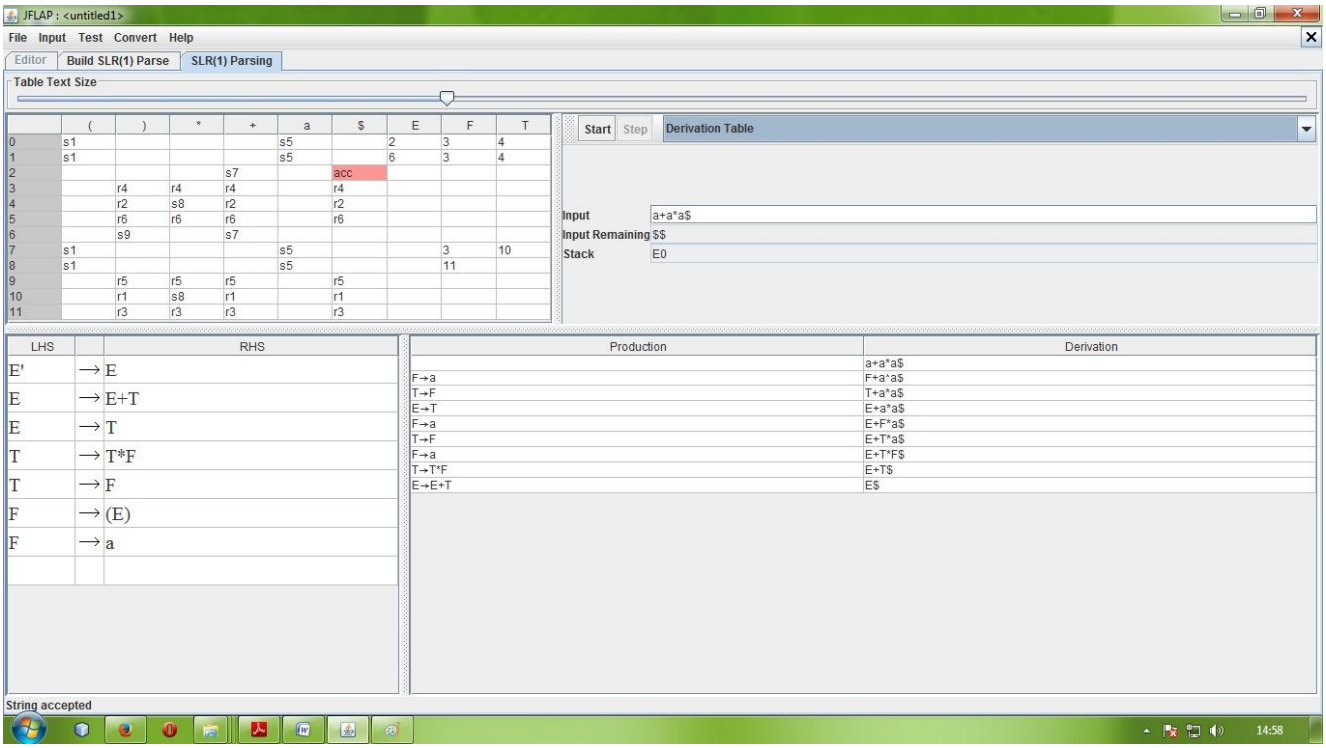


Fig 5

V. CONCLUSION

In this paper, we presented JFLAP, a tool for interactive learning of formal languages and automata theory of context free grammars. The objective of the tool is to make the learning process entertaining and simple for students, while lessening the burden of problem and example preparation for the lecturers. Our experiences from the development of *Automata Simulator* [9] and *SoftLab* [10]. A further step in improving the tool and making it classroom-ready is implementation of conversions between basic models of finite state machines (NFA epsilon, NFA, and DFA), as well as implementation of DFA minimization.

VI. REFERENCES

[1] Ivan Budiselic, Sinisa Srbljic and Miroslav Popovic, "*RegExpert*: A Tool for Visualization of Regular Expressions", The IEEE International Conference on "Computer as a Tool", EUROCON 2007.

[2] S. Srbljic, *Compiler Design 1: Introduction to Theory of Formal Languages, Automata, and Grammars*, (original title in Croatian: Jezicni procesori 1: uvod u teoriju formalnih jezika, automata igramatika"), 2nd Edition, Element, Zagreb, 2002, pp. 46-50.

[3] S. Wolfram, *The Mathematica Book*, 5th Edition, Wolfram Media, 2003.

[4] Pemmaraju and S. Skiena, *Computational DiscreteMathematics: Combinatorics and Graph Theory with Mathematica*, Cambridge University Press, 2003.

[5] B. aneva and D. Thiébaut, "Sorting algorithms," Smith College, http://maven.smith.edu/~thiebaut/java/sort/demo.html.

[6] A. Jacobsen, "Data-link network protocol simulation," University of Birmingham, School of Computer Science, http://www.cs.bham.ac.uk/~gkt/Teaching/SEM335/dlsim/Simulation.html.

[7] B. W. Watson, "The design and implementation of the FIRE engine: a C++ toolkit for finite automata and regular expressions," *Computing Science Note 94/22*, Eindhoven University of Technology, Netherlands, 1994.

[8] Grail+ Project homepage, http://www.csd.uwo.ca/Research/grail/index.html.

[9] S. H. Rodger and T. W. Finley, *JFLAP: An Interactive Formal Languages and Automata Package*, Jones & Bartlett Publishers, Sudbury, MA, 2006.

[10] Skuliber, S. Srbljic, and I. Crkvenac, "Using interactivity in computer-facilitated learning for efficient comprehension of mathematical abstractions," *Proceedings of the EUROCON 2001*, International Conference on Trends in Communication, Bratislava, Slovak Republic, July, 2001, vol. 2/2, pp. 278-281.

11] Skuliber, S. Srbljic, and A. Milanovic, "Extending the textbook: a distributed tool for learning automata theory fundamentals," *Proceedings of the 9th IEEE International Conference on Electronics, Circuits and Systems* (*ICECS 2002*), Dubrovnik, Croatia, September, 2002.