Proceedings of the Pakistan Academy of Sciences:
A. Physical and Computational Sciences 55 (2): 55–67 (2018)

ISSN: 2518-4245 (print), 2518-4253 (online)

Pakistan Academy of Sciences

Research Article

Didactic Strategy for Learning Theory of Automata & Formal Languages

Muhammad Shumail Naveed[1*], and Muhammad Sarim[2]

[1]Department of Computer Science, University of Balochistan, Quetta, Pakistan
[2]Department of Computer Science, Federal Urdu University of Arts, Science & Technology, Karachi, Pakistan

Abstract: Formal languages and automata theory have a strong association with the core of information in the area of computer science. However, the courses on formal languages and automata theory is a challenging task and students generally do not find these courses very attractive and experience intricacy and impediment in learning the concepts. These intricacies stem from the difficult and unusual abstract concepts and the essential mathematical background. This paper presents a didactic strategy to simplify the hardness of the courses on formal languages and automata theory and aids to increase the interest and commitment of students in these courses. The proposed strategy supports a more imperative learning of the topics in formal languages and automata theory in an effective and fruitful way. The strategy initially evaluated and primary results are quite encouraging.

Keywords: Learning formal languages and automata theory, theory of computing, students' disappointment, simulation tools, pair programming, education.

1. INTRODUCTION

The formal languages and automata theory (FLAT) is a core of computer science curriculum [1, 2], and usually offered at undergraduate level. FLAT has a significant role in different areas, particularly in the theory of computation, artificial intelligence and compiler construction. However, many students usually find these courses tedious and complex [3, 4, 5]. Students usually find FLAT courses to be archaic and cannot associate their topics to other courses in the computer science curriculum or computer applications [6], and therefore students taking a course on FLAT be likely to be unenthusiastic and are unmotivated. Verma [7] argues that this frequently leads to student disappointment and high dropout rate in FLAT courses than other courses.

The FLAT courses are mostly introduced without the use of computers and generally do not include any programming [8]. A very few FLAT courses comprised of practical project such as the development of lexical analyser for a compiler. In fact, the induction of practical aspects of the FLAT courses is infrequent and typically embraced in courses that include further topics like parsing.

Normally the conventional approaches are used to introduce the FLAT courses, which are generally based on the integration of lectures with tutorials. Conventional chalk-and-board approaches are followed to teach students the different concepts of FLAT courses. With tutorials the students study the FLAT concepts by working through exercises which are presented for assessment.

It has been observed that a large majority of students is less motivated in understanding the concepts of FLAT courses. The lack of interest is not only due to the difficulties of the topics themselves but also due to the facts that the orientation and method of teaching many of the topics in FLAT courses are not towards computer science but biased towards mathematics. In most of the computer science undergraduate curriculum a Discrete Mathematics is a perquisite of FLAT

Received, January 2017; Accepted, June 2018
* Corresponding Author: Muhammad Shumail Naveed; mshumailn@gmail.com

courses.

In order to increase the learning of students in FLAT courses, this paper presents a didactic strategy based on an amalgamation of existing techniques with some new concepts. The central aim of the proposed strategy is to overcome the complexity and abstractness of topics by turning the FLAT course more computer-oriented and interesting.

The paper is organized as follows. In section 2 the related work conducted in the support of FLAT courses is discussed. The main facets of a proposed strategy are described in section 3. The results of the initial evaluation and discussion of a proposed strategy are illustrated in section 4. Finally, section 5 describes the conclusion.

2. RELATED WORK

A lot of work has been conducted to increase the learning of FLAT courses. In [9], a strategy based on constructivist approach is defined by integrating different teaching approaches to increase the interest of students and the initial results of its application are quite satisfactory. In [10], D'antoni et al. elucidate automatic feedback in the learning of deterministic finite automata constructions by analyzing the binary and counterexample-based feedback. The initial response of automatic feedback is quite encouraging.

Moura and Dias [11] introduced L-FLAT, a Logtalk Toolkit for teaching formal languages and automata theory. It supports regular expressions, finite state automata, context-free grammar, pushdown automata and Turing machines. L-FLAT units are defined using object-oriented aspects of Logtalk. The significance of L-FLAT in pedagogical environment is increased by bracing Mooshak, a web-oriented application which supports automatic ranking of presented programs.

The FSM is described to present students with the prospect to work and verify the designs by using regular expressions, state machines and grammars [12]. In [13], the use of pen-based computing to promote the learning in formal languages and automata theory is proposed. Dol [14] suggests the use of Think-Pair-Share, which is a cooperative learning strategy to increase the students learning about the course.

Educational software tools are used to augment teaching strategies and particularly software simulators present association between theory and practice. Several educational software tools have been designed to support the simple and interactive learning of formal languages and automata theory. SELFA (Software for Learning Formal languages and Automata theory) is an educational simulation tool, developed to improve the standard of teaching in formal language and theory of automata courses [15]. The endeavour of SELFA tool is to make it simple to teach the concepts of the subject, whose level of abstraction make the process complex.

Thoth is technique which helps in teaching FLAT courses [16]. It can simulate push-down automata, Turing machines and other classical concepts like regular expressions, finite automata and context-free grammars. The tool is specially designed to allow easy interaction with different concepts of FLAT courses and allows students to easily experiment with different designs and observe the step-by-step evaluation of algorithms. Thoth has a simple and friendly user interface which allows the easy and rapid design of the automata. The interface of Thoth is available in English, German, French and Spanish.

FLUTE (Formal Languages and aUTomata Environment) is one of an important effort made to simplify the learning of formal languages and automata [17]. Basically FLUTE is an intelligent tutoring system that helps students in learning individually about FLAT.

Nóbrega et al. [18] described to use a Semantic Wiki as a tool to help FLAT course by associating tools like JFLAP. Its main objective is to increase the engagement of students in FLAT course.

JFLAP is one of an educational tools used in FLAT courses. Several studies [19, 20, 21] have been conducted on the use of JFLAP in FLAT courses and reported significant improvement in controlling the hardness of these courses. Jarvis and Lucas [22] have modified JFLAP through which it is possible to develop Java programs that change the actual automaton itself and found it very useful in increasing the student's capability to understand fundamental concepts like Church-Turing thesis and the undecidability.

Automata Tutor is an online tool that aids students to comprehend essential notions in the theory of computation, such as regular expressions and finite automata. It also provides feedback when students define wrong solutions, and also supports instructors in organizing large classes by grading homework assignments. The tool has been used in several universities by many students. The Automata Tutor is analyzed and it has been reported that students were more engaged with the course content by interacting with the tool [23]. Similarly, the use of a tool increased average grade on the assignments of homework and teachers enjoyed the tool.

de Souza et al. [24] introduced a teaching-learning methodology to aid the formalism model construction about FLAT. According to that methodology, for every topic of FLAT, the instructor must recommend a simulator development as classwork or homework to help the student in formalism learning. A simulator based Multi-Formalism Modeling is developed and include icon-based interface and the initial results of the proposed methodology are quite positive.

A Combined Methodology based on simulator development is introduced to provide the knowledge required in the theoretical computer science area during the classes of FLAT, compilers and computer science theory [25]. The methodology was statistically analysed and found very effective. Neeman described the use of software testing techniques to elucidate some concepts of the theory of automata. The method utilizes equivalence partitioning and provides a guide for the various concepts of automata theory [26].

3. DESIGN & METHOD

The topical research on FLAT courses including the studies referred in the previous sections demonstrate that the abstract nature of concepts and notations involved in the course, hard background of mathematics, conventional style of teaching and student's perception on the irrelevancy of a FLAT course with the computer science are the contributing factors behind the hardness of FLAT course.

The contemporary solutions for FLAT courses usually addressed the single or a few issues whereas the didactic strategy presented in this paper aims to overcome the intrinsic complexity of FLAT by considering the multiple issues involved in the learning of formal languages and theory of automata and perhaps this is a main novelty of the proposed didactic strategy. Principally, the proposed strategy is based on five principles.

3.1 Illustration of the Course in Linking with Computer Science History

Adding the details of the historical development of the theory of computing in conjunction with the topics of the formal language and automata theory has recognized very useful and encouraging for students [9], and therefore it is a first principle of the proposed strategy. The course on FLAT is not based on history of computer science; but it can be augmented by adding details related to the historical background in which the topic appeared as a new discipline. Including historical information would help the students in discovering how several concepts developed over time, and how research works and contribute to the expansion of a new field. For example, when introducing the concept of procedure and Turing-Church thesis it is useful to describe significant essentials about Alonzo Church and Alan Mathison Turing. So, the students come to know that both Alan Mathison Turing and Stephen Cole Kleene did their doctorate under the direction of Alonzo Church and therefore their research dimensions were following similar targets. Avram Noam Chomsky initiated the area of formal language theory in 1950s to define formal description of the structure of natural languages [27], so during introducing the types of grammar and particularly the context-free grammar it is fruitful to describe some pertinent work and development made by Noam Chomsky in the area of linguistics. Different methods like biographical notes, conventional lectures and videos may be used to describe the historical development; however, it is important to consider the cognitive load while including these details.

3.2 Pair Programming

Students usually encounter anxiety when they begin to learn a course on FLAT and therefore students are motivated and encouraged to work in a pair. Working in a pair is a second principle of the proposed strategy. Pair programming has been

largely utilized in computer science education because of the benefits it provides to students [28].

Pair programming is an active research discipline and one of a useful area of Extreme Programming. In this method, two programmers work together at one computer on the same task. A person who holds the keyword is called the *driver* and the person who sits alongside the driver is called *navigator*. The basic duty of a driver is to develop the code. The navigator reviews the code and looks for possible errors that a driver leaves the code by mistake.

During FLAT course, the students are motivated to work in a pair as it is favourable for students and increased their confidence and performance. During problem solving, the driver develops the logic and solves the problem; while the other, the navigator analysed the logic. The students switch roles frequently.

3.3 Induction of Software Tools for Active Learning

The conventional chalk-and-board education style used to teach the concepts in FLAT courses is hard for students to learn the different concepts and proofs [29, 30], and therefore the pedagogy of FLAT courses is shifting from pencil and paper environment to a technological environment which use educational software to work with different concepts in the course [19]. Morazan and Antunez in [12] described that FLAT courses should be taught by using tools to define computation. In many areas of computer science, the visualization tools, simulators and other educational software are productively used to illustrate and visualize the abstract concepts. The third principle of a proposed strategy is to use different software tools to teach the FLAT courses. The use of suitable software would allow the students of FLAT courses to work and experiment the concepts that would be hard and complex to do on paper, and to get immediate response to the problem solving. Chudá and Rodina in [31] reported that the use of software tools like simulators in FLAT courses can significantly affect the pedagogical process.

Large varieties of tools have been developed to stimulate students' interest and help them in comprehending the FLAT concepts. These tools help students to easily learn and experiment the concepts which are usually tedious to do on a paper. Many visualization tools allow animation of different constructs and proofs and most of these tools are freely and widely available. Simulation of automata for educational purposes is itself a significant area in computer science education research [32]. Chesñevar et al. [1], discussed the main aspects of different educational software used for teaching FLAT and categorized these tools into multi-purpose tools and single-purpose tools. The multi-purpose tools and single-purpose tools are both helpful in the active learning of FLAT courses.

RegeXeX (Regular expression exercises) is one of a useful tool that can be used in active learning of regular expressions [33]. It provides a collection of exercises controlled by a system and provides advice to students on the precision of solutions. RegeXeX is also very useful during lectures to describe how the regular expression is generated.

CAVE (Constructive Algorithm Visualization Environment) is a pedagogical help to support instructors to describe the construction of deterministic finite automata and combining DFAs using different operators [1]. FSME (Finite State Machine Explorer) provides an environment that allows the students to construct finite state machines and verify them on different inputs. It also provides the conversion between equivalent classes of finite state machines.

MACH0 is a graphical simulation tool of deterministic finite state automata. It accepts the definition of deterministic finite automaton and displays its transition diagram and simulate the working of any input string. It uses colourful animation to increase learning and could be used to enhance the learning of FLAT courses.

Mealy and Moore machines are the essential components of FLAT courses and these machines can be introduced with TAGS (Transducer Automata Graphical Simulator) which allows defining and running these machines [34]. It also allows the rewinding of simulation. The tool also allows stepwise or continuous simulation.

The FSA Simulator is a program developed to enable students to work with finite state automata [35]. It also allows comparing the languages of two automata. FSA simulator is one of a popular tool and could be used in FLAT courses. Proof

Checker is a graphical environment that allows students to generate deterministic finite automaton and verify its correctness [36]. Proof Checker is an elegant tool that may be used to aid the students in comprehending the FLAT courses.

Formal Language and Automata Package (FLAP) is a simulation tool to design and simulate finite automata, pushdown automata and Turing machines [37]. It also supports numerous deterministic and nondeterministic dialects of these automata. The tool allows the simple and easy generation of automata by simply clicking and dragging of a mouse. JFLAP is free and interactive educational software to learn the topics in formal languages and automata theory. It is used to support the FLAT contents, including regular languages, context free languages, unrestricted grammars, pumping lemmas and Turing machines and recognized as a useful tool to reduce the gap between students and FLAT contents [5].

The Abstract Machine Simulator is another tool designed to help student to comprehend finite state automata and transducers [38]. It accepts the description of an automaton in tabular form. Text based and graphical simulation modules are available in the system.

Language Emulator is another powerful tool for the simulation of finite state automata, Mealy machine and Moore machines [39]. The tool is available in English and Portuguese. It accepts the description of the automaton and simulates its behaviour. It also allows the conversion of NFA to DFA, minimization of finite automaton and conversion between Mealy machine and a Moore Machine.

The Turing Machine Simulator is a tool for the simulation of Turing machine [40]. It allows the stepwise simulation of the Turing machine with descriptive comments. Turing Machine Simulator was originally developed by McFall and Dershem to design and simulate finite state automata and Turing machines [41]. The tool allows both nondeterministic and deterministic automaton. It also permits the use of sub-machines in the definition of an automaton. Turing's world is another tool developed to design and simulates Turing machine and very effective for pedagogical purpose. It also supports sub-machine for complex Turing machines.

Turing Building Block is a tool developed by Luce and Rodger to design and simulate Turing machines [42]. It also supports modular design. The tool includes a graphical editor for interactive and visual designing of a machine.

Interactive Pushdown Automata Animation is a tool developed by McDonald to design and simulate pushdown automata [43]. The tool allows instantaneous and stepwise simulation. It also supports high graphics and very effective for learning. Finite State Automata Simulator is another tool to design the deterministic and nondeterministic finite state automata [44]. The tool allows instantaneous and stepwise simulation.

Finite State Machine Simulator [45] is a tool developed to design and simulates deterministic and nondeterministic finite automata in a visual manner. It also supports the conversions of a nondeterministic finite state automaton to an equivalent deterministic finite state automaton and also supports the minimizing of a finite state automaton.

Java Finite Automata Simulation Tool is a very helpful tool to design and simulate finite state automata, pushdown automata, Turing machines and other forms of automata [46]. The tool also supports nondeterministic and deterministic machines. SimStudio is a simulation tool developed for finite state automata, pushdown automata and Turing machines [31]. The tool also supports nondeterministic and deterministic machines.

Apart from the above tools, there are other tools like FAdo, IPAA (Interaction Pushdown Automata Animation), Visual Turing, Minerva, DEM, JCT and A to CC which are highly amenable for comprehending the FLAT courses. Though there are several tools for learning FLAT courses and it's possible to use multiple tools to introduce the concepts; however it is recommended to use a single or minimum tools to cover the topics in a course otherwise it would increase the learning load.

3.4 Computational View of a Course

The mathematical nature of FLAT makes the courses more abstract and difficult for students

[7, 39]. The students of computer science undergraduate programs usually have no strong background of mathematics and consequently they have little interest and motivation in FLAT courses. However, it is possible to increase the interest of students and reduce the hardness by introducing the FLAT courses in the context and application of computer science and this is the central theme of a fourth principle of a proposed didactic strategy. Devedzic et al. [47], argued that interest of students can be increased by illustrating them where they can implement the knowledge they have acquired. The applications helpful in the realization of the fourth principle of a proposed didactic strategy include formal verification and logic programming [7], programming languages, query languages, data definition languages, regular expressions-based text searches and communication protocols [48]. Finite state automata and regular expression can be introduced to describe the lexical analyser, nondeterministic finite automaton can be used to model a critical processing with a binary semaphore [49]; finite automaton may be used in user interface design; grammar can be used in the definition of word processors and programming languages; pushdown automaton can be used to describe the syntax analyser and halting problem can be introduced in connection with antivirus problems.

Model-based testing is a central element of contemporary test automation. In this technique a software is evaluated by analysing the run time behaviour against predications defined by a proper specification. The well-established techniques use finite state machines as a foundation to select test inputs [50].

Partial order reduction is a technique used for concurrent asynchronous systems and use model checking [51]. Model checking is a technique to verify sequential circuit diagrams and communication protocols. In this system, proportional temporal logic is conventionally used to define the specifications, and the system is modelled as a state-transition graph.

3.5 Use of Heuristics in Problem Solving

Construction of different machines and logics in FLAT courses are usually non-trivial for beginners because there is no mathematical formula or well-defined recipe for their construction, and every problem has its own requirement and entails a different approach for its solution and consequently it is usually very hard for beginners to apply the knowledge of one problem in the direct solution of other problems. However, it is possible to define some heuristics as shortcuts to ease the construction of automata and other problems. Use of heuristics in FLAT courses is a fifth principle of the proposed strategy. These heuristics work as an educated guess or a rule of thumb to understand and define solutions more quickly.

The heuristics can be easily identified with experience, observation and judgment and every course instructor needs to establish his own heuristics for FLAT courses and therefore the paper does not define any exhaustive list of heuristics. However, the authors have established and experienced the following heuristics for FLAT courses.

1. Although the finite state automaton can be constructed in any arbitrary fashion; however, it is better to construct the automaton in top-down and left-right manner as shown in the example illustrated in Fig. 1.

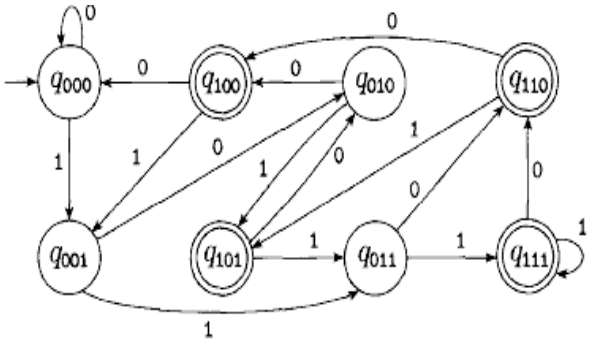

Fig. 1. Sample finite automaton for heuristic 1 [52]

   This heuristic not only increased the understandability of a finite automaton, but also helps students in tracing the strings.

2. Any language defined by a finite automaton can be constructed with the regular expression, and the regular expressions always generate the regular languages. Therefore, the shape of transition graph of an automaton should follow some regular or symmetrical pattern. Although the structure of transition graph solely depends on the structure of an underlying language, yet it is better to construct the transition graph in a form of rectangle, square, pyramid, diamond,

oval or their combination. As an example, consider a finite automaton shown in Fig. 2.

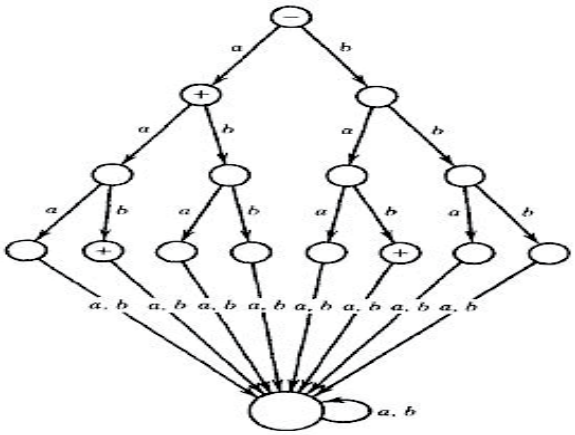

Fig. 2. Sample automaton for heuristic 2 [53]

Above automaton only accepts the strings *a*, *aab*, and *bab* and it is apparent that it follows one of a prescribed shape.

3. If an empty string is a part of a regular language, then the initial state of its deterministic finite automaton should be a final state or one of a final state of a machine. As an example, consider an FA (shown in Fig. 3) developed over the alphabet Σ = {0, 1}, accepting all strings with an even number of 0's and even number of 1's:

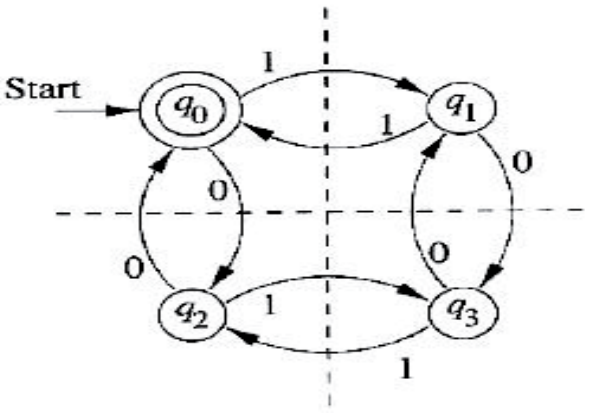

Fig. 3. Sample finite automaton for heuristic 3 [54]

In this example, the empty string is a valid string in a language; therefore, the initial state is also a final state of a machine. The same heuristic can be used in a definition of a pushdown automaton. If a context-free language includes empty string, then its respective pushdown automaton usually contains an empty move from the first input state which may check the stack and switch to the acceptance of a string. As an example, consider the Fig. 4.

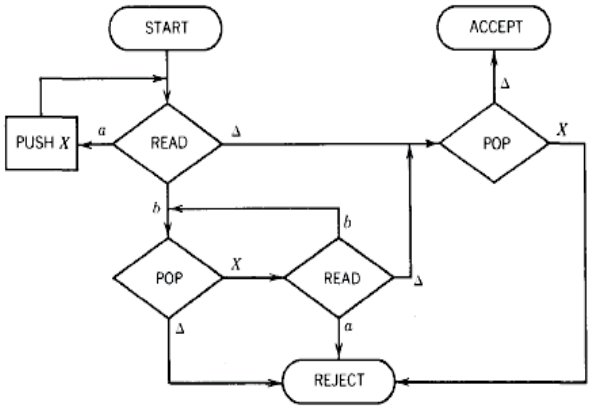

Fig. 4. Sample pushdown automaton for heuristic 3 [53]

This pushdown recognized a language a^nb^n. The language includes an empty string and the first input state of the pushdown automaton check the ε-input and accepts it after verifying the stack.

4. If a regular language is finite, then its' respective regular expression never contains a closure and similarly the equivalent finite automaton never contains a loop. Conversely, if a language is infinite, then its respective regular expression and its equivalent finite automaton should contain the loop(s) which could either be a direct or indirect.

Consider a language {aa, aabb}*{b}. The language is infinite and therefore its regular expression (i.e., $(aa+aab)^*b$)) contains a closure and the respective finite automaton contains a loop as shown in Fig. 5.

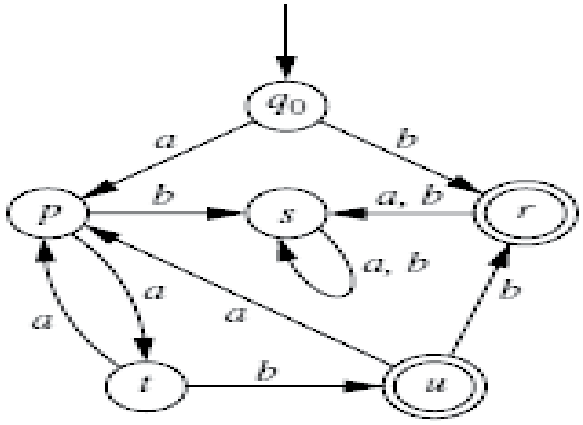

Fig. 5. Finite automaton involving indirect loop [55]

In the definition and use of heuristics, it is important to realize that heuristic is simply an educated guess, but not a firm rule so the exceptions are always possible.

4. PRELIMINARY EVALUATION AND DISCUSSION

A small study is conducted to determine whether the proposed solution is justifiable and fruitful to reduce the complexity of FLAT course and increase the interest of students. The study comprised of two parts. In the first part, an online survey was conducted in which the undergraduate computer science students of Pakistan who have taken a course on FLAT were asked the following question:

"*Formal language and automata is one of a useful and influential subject/course of computer science*"

The respondent can reply with 5-item Likert Scale. During the survey, 183 responses are received from the different regions of Pakistan. The number of respondents participated from the different regions of Pakistan and shown in Fig. 6.

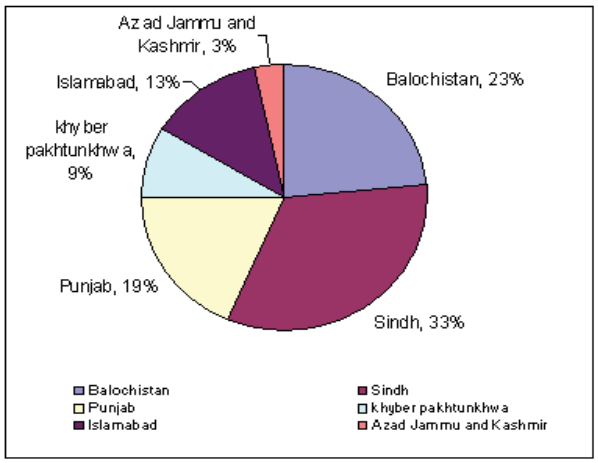

Fig. 6. Geographical locations of respondents

During online survey, 23% response is received from Balochistan, 33% from Sindh, 19% from Punjab, 9% from Khyber Pakhtunkhwa and 3% from Azad Jammu & Kashmir.

The region wise responses received from the partcipants are shown in Fig. 7.

It can be seen that a large majority of students do not recognize the FLAT course, as a useful and influential subject of computer science and the large majority of students disagree with the usefulness of FLAT course. The same information can be better understood with the Fig. 8.

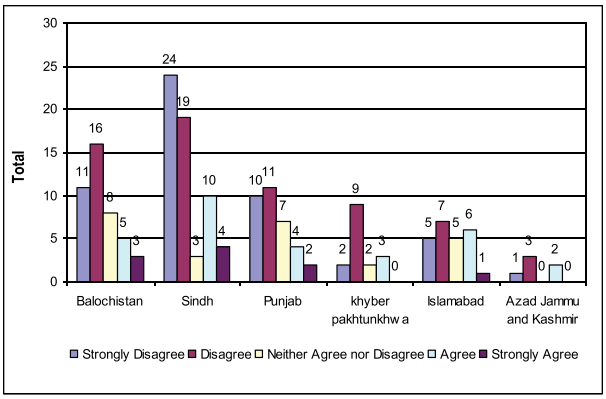

Fig. 7. Region wise response of participants

The feedback received from the students indicates that 24% of students strongly disagreed and 39% are disagreed with the useful and influence of FLAT course, whereas 20% agreed, 4% strongly agreed and 13% neither agreed nor disagreed with the significance of FLAT courses.

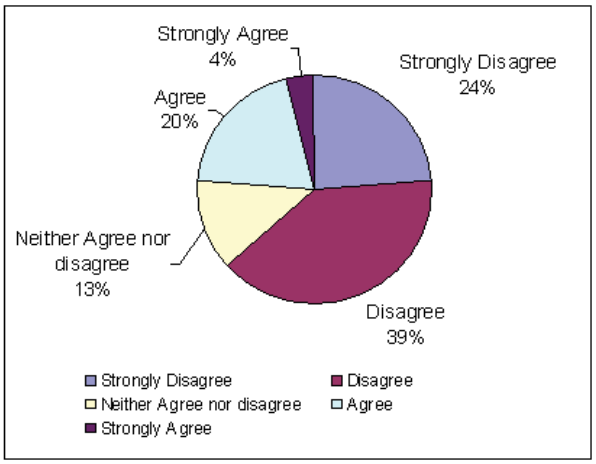

Fig. 8. Percent wise illustration of feedback

The overall response received from the survey suggest that undergraduate computer science students of Pakistan do not recognize the FLAT as a useful and persuasive course which implied the need of a didactic strategy that could increase the performance and motivation of students. Therefore, as a second part of an evaluation a small study is conducted. During the study, 64 students in the undergraduate computer science program who have already studied different subjects including: introduction to computing, programming

Table 1. Results of evaluation

S. No.	Group	Students	Pass	Fail	Mean	Std. Deviation	Std. Error
1	Control	32	17	15	50.09	20.31	3.59
2	Treatment	32	20	12	58.81	19.30	3.41

fundamentals, object-oriented programming and discrete structures in previous semesters are selected and randomly divided into a control group and treatment group. A same instructor to both group of studies offers a course on formal language and automata theory of three credit hours.

During the study, a conventional approach is followed for the control group, while the proposed strategy described in the previous section is utilized in a treatment group. For the realization of first principle, the historical notes outlined in the previous section and many other historical notes are followed to introduce the course. Traditional lectures with multimedia-oriented lectures are delivered to link the FLAT course with the computer science history.

During problem solving and class activities, the students were motivated to work in a pair. It is suggested that different concepts in FLAT courses can be introduced in the context of a particular programming language and it is also widely recognized that automata play a significant role in the design of the compiler [9,54] . So, during the evaluation of proposed strategy the FLAT course is introduced in the context of programming language and compiler construction. A small grammar of C-type language is developed to introduce the context-free grammar and its recognition is described by introducing pushdown automaton and parsing. Similarly, the lexical aspects of a designed programming language are described by the use of finite automata and regular expressions.

JFLAP is mainly used to introduce the FLAT course to treatment group; however a slight concept of RegeXeX is also introduced to the students of treatment group. During the course, students of treatment group are introduced with different heuristics which help them to understand the problems and generate their solutions.

After the completion of course, the control and treatment groups are internally evaluated in this study. Table 1 shows the summary of the results.

The pass rate in the control group is 53.13 whereas 62.5 in the treatment group. Fig. 9 illustrates the box plots of marks obtained by both groups.

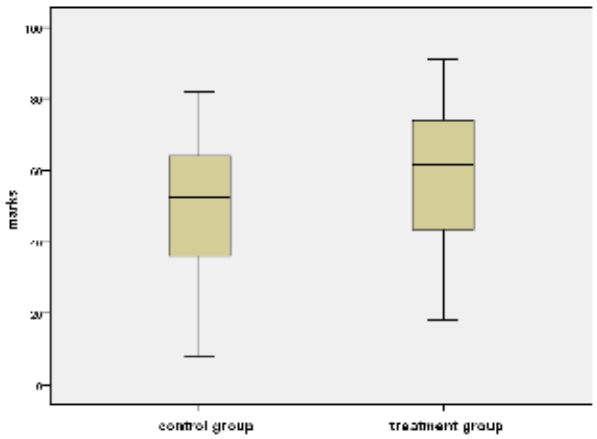

Fig. 9. Box plots of marks

The quartiles of the box plots show that the performance of students in the treatment group is much better than the students of a control group.

The significance of a proposed didactic strategy is analysed by applying the independent sample t-test on the marks secured by the students, and the calculated t-value is 1.76, and p-value is .042, so the result is statistically significant at $p < .05$.

Maintaining the interest of students is one of a main challenge in FLAT courses. So, in order to identify whether the proposed didactic strategy increased the interest and motivation of students, the following question has been asked before the exams to the both groups of students:

"*I am enthusiastic about this course*"

The students can reply on Likert scale ranging from 1 (strongly disagree) to 5 (strongly agree). The feedback received from the students is shown in Fig. 10.

Fig. 10 shows that interest and motivation level of students in the treatment group is much higher than the control group.

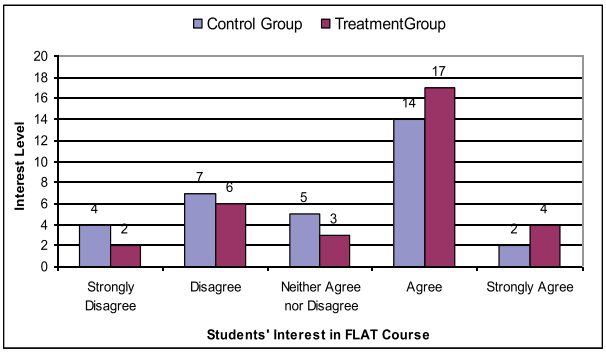

Fig. 10. Students interest in FLAT course

Overall, the results acquired from the initial evaluation suggest that the proposed didactic strategy is useful to simplify the hardness of the course on formal languages and automata theory and useful to increase the interest of students. The results of evaluation naturally implied that the abstract nature of the notions involved in the FLAT course, traditional style of teaching and student's perception on the irrelevancy of a FLAT course with the computer science are the major factors behind the hardness of FLAT course.

The results obtained from the initial evaluation of the proposed strategy are quite encouraging. The comparison of a proposed didactic strategy and the other strategies is shown in Table 2.

The results shown in Table 2 indicate that the proposed study is quite fruitful and supportive in increasing the performance of students in FLAT courses and thereby comparable to the other didactic strategies.

5. CONCLUSION

The courses on formal languages and theory of automata present various issues for the instructors. Top among these is the student disappointment and high dropout due to the abstract nature of a course and material, students' perception on the irrelevancy of FLAT course in computer science and the firm background in mathematics. This paper presents a didactics strategy and reports on its initial evaluation. In its current state, the strategy is based on five

Table 2. Comparison of didactic strategies

S. No.	Study	Method	Results
1	Pillay [2]	No practical study is reported	Proposed model was judged on observation and on projected benefits
2	Chesnevar et al. [9]	No practical study is reported	Proposed model was analyzed on observation and on projected benefits
3	D'antoni et al. [10]	Detailed study is conducted	Statistically significant, $p < 0.005$
4	Moura, P. & A.M. Dias [11]	No study is reported	Proposed model was analyzed on observation and on projected benefits
5	Morazan and Antunez [12]	No study is reported	Proposed model was analyzed on observation and on projected benefits
6	Castro-Schez et al. [15]	Two studies are conducted	Average pass rate of students in two studies is 61%
7	Nóbrega, G., F. Lima & D. Freire [18]	No study is reported	Proposed model was analyzed on observation and on projected benefits
8	Neeman [26]	Survey based questions	60% students understood the major concepts of FLAT.
9	Singh and Isah [27]	No study is reported	Proposed model was analyzed on observation and on projected benefits
10	Brown and Hardisty [33]	Classroom experience	Mean score of control group is 30. T-test was statistically significant with a p-value of 0.034
11	Proposed Strategy	Practical study is conducted	Average pass rate of students is 62.5 and results are statistically significant with a p-value of 0.42

rational principles. The proposed strategy initially applied to a small group of students and the results are quite satisfactory. The initial results suggest that illustration of FLAT course in linking with computer science history, motivating the students to work in a pair, use of software tools during lectures, introducing the computational view of a course and the use of heuristics for problem solving can overcome the intrinsic hardness of FLAT courses and may increase the interest of students. Further study is underway in four dimensions i) definition of viable methods for the pairing of students ii) comparative analysis of FLAT tools iii) analysing the significance of constructivism in FLAT courses iv) evaluation of proposed didactic strategy on a large group of students.

6. ACKNOWLEDGMENTS

The authors are grateful to the Department of Computer for the support. The authors would also like to thank all the students who participated in the study.

7. REFERENCES

1. Chesñevar, C.I., M.L. Cobo. & W. Yurcik. Using theoretical computer simulators for formal languages and automata theory. *ACM SIGCSE Bulletin* 35: 33-37 (2003).
2. Pillay, N. Teaching the theory of formal languages and automata in the computer science undergraduate curriculum. *South African Computer Journal* 12: 87-94 (2008).
3. Pillay, N. Learning difficulties experienced by students in a course on formal languages and automata theory. *ACM SIGCSE Bulletin* 41: 48-52 (2010).
4. Lakshmi. & R. Sukumaran. Use of ICT in learning theory of computation: An experimental study. In: *IEEE International Conference in MOOC Innovation and Technology in Education*, Jaipur, India. p. 109-113 (2013).
5. Paul, J. Using jFlap to engage students and improve learning of computer science theory: tutorial presentation. *Journal of Computing Sciences in Colleges* 31: 145-148 (2015).
6. Korte, L., S. Anderson, H. Pain. & J. Good . Learning by game-building: a novel approach to theoretical computer science education. *ACM SIGCSE Bulletin* 39: 53-57 (2007).
7. Verma, R. M. A visual and interactive automata theory course emphasizing breadth of automata. *ACM SIGCSE Bulletin* 37: 325-329 (2005).
8. Moreira, N. & R. Reis. Interactive manipulation of regular objects with fado. *ACM SIGCSE Bulletin* 37: 335-339 (2005).
9. Chesnevar, C.I., M.P. Gonzalez. & A.G. Maguitman. Didactic strategies for promoting significant learning in formal languages and automata theory. *ACM SIGCSE Bulletin* 36: 7-11 (2004).
10. D'antoni, L., D. Kini, R. Alur, S. Gulwani, M. Viswanathan. & B. Hartmann. How can automatic feedback help students construct automata?, *ACM Transactions on Computer-Human Interaction* 22: (2015).
11. Moura, P. & A.M. Dias. L-flat: Logtalk toolkit for formal languages and automata theory. In: *Proceedings of the 11th International Colloquium on Implementation of Constraint Logic Programming Systems,* Kentucky, USA, (2011).
12. Morazan, M.T. & R. Antunez. Functional automata formal languages for computer science students. In: *Proceedings of Trends in Functional Programming in Education,* Soesterberg, The Netherlands, 19-32 (2014).
13. Berque, D., D.K. Johnson. & L. Jovanovic. Teaching theory of computation using pen-based computers and an electronic whiteboard. *ACM SIGCSE Bulletin* 33: 169-172 (2001).
14. Dol, S.M. Tps (think-pair-share): An active learning strategy to teach theory of computation course. *International Journal of Educational Research and Technology* 5: 62-67 (2014).
15. Castro-Schez, J. J., E.E. Castillo, J. Hortolano. & A. Rodriguez. Designing and using software tools for educational purposes: Flat, a case study. *Transactions on Education* 52: 66-74 (2009).
16. García-Osorio, C., I. Mediavilla-Sáiz, J. Jimeno-Visitación. & N. García-Pedrajas. Teaching push-down automata and turing machines. *ACM SIGCSE Bulletin* 40: 316 (2008).
17. Devedzic, V., J. Debenham. & D. Popovic. Teaching formal languages by an intelligent tutoring system. *Educational Technology & Society* 3: 36-49 (2000).
18. Nóbrega, G., F. Lima & D. Freire. Integrating the semantic wiki approach to face to face courses. In: *World Conference on Computers in Education,* Bento Gonçalves, Brazil. p. 1-19 (2009).
19. Rodger, S. H., J. Lim. & S. Reading. Increasing interaction and support in the formal languages and automata theory course. *ACM SIGCSE Bulletin* 39: 58-62 (2007).
20. Cavalcante, R., T. F. Cornell. & S. H. Rodger. A visual and interactive automata theory course with jflap 4.0. *ACM SIGCSE Bulletin* 36: 140-144 (2004).
21. Rodger, S. H., E. Wiebe, K. M. Lee, C. Morgan, K. Omar. & J. Su. Increasing engagement in automata theory with jflap. *ACM SIGCSE Bulletin* 4: 403-407 (2009).
22. Jarvis, J., J.M. Lucas. Incorporating transformations

into jflap for enhanced understanding of automata. *ACM SIGCSE Bulletin* 40: 14-18 (2008).

23. D'Antoniy, L., M. Weavery, A. Weinertz & R. Alury. Automata tutor and what we learned from building an online teaching tool. *Bulletin of EATCS* 3; (2015).
24. de Souza, G.S., C. Olivete, R.C. Correia. & R.E. Garcia. Teaching-learning methodology for formal languages and automata theory. In: *IEEE Frontiers in Education Conference,* El Paso, USA. p. 1-7 (2015).
25. de Souza, G.S., G.P.H. Gomes, R.C.M. Correia, C. Olivete, D.M. Eler. & R.E. Garcia. Combined methodology for theoretical computing. In: *Frontiers in Education Conference*, Erie, USA. p. 1-7 (2016).
26. Neeman, A. Buy one get one free: automata theory concepts through software test. *Journal of Computing Sciences in Colleges* 31: 90-96 (2016).
27. Singh, D. & A.I. Isah. An outline of the development of the theory of formal languages. *International Journal of Latest Trends in Computing* 5: 172-181 (2014).
28. Myller, N., R. Bednarik, E. Sutinen. & M. Ben-Ari. Extending the engagement taxonomy: Software visualization and collaborative learning. *ACM Transactions on Computing Education* 9: 7:1-7:27 (2009).
29. McDonald, J. Interactive pushdown automata animation. *ACM SIGCSE Bulletin* 34: 376-380 (2002).
30. Chud'a, D. Visualization in education of theoretical computer science. In: *Proceedings of the 2007 international conference on Computer systems and technologies,*; Rousse, Bulgaria. p. 84:1-84:6 (2007).
31. Chud'a, D. & D. Rodina. Automata simulator. In: *Proceedings of the 11th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing on International Conference on Computer Systems and Technologies*, Sofia, Bulgaria. p. 394-399 (2010).
32. Chakraborty, P., P.C. Saxena. & C.P. Katti. Fifty years of automata simulation: a review. *ACM Inroads*; 2: 59-70 (2011).
33. Brown, C.W. & E.A. Hardisty. Regexex: an interactive system providing regular expression exercises. *ACM SIGCSE Bulletin* 39: 445-449 (2007).
34. Esmoris, A., C.I. Chesnevar & M.P. Gonzalez. Tags: a software tool for simulating transducer automata. *International journal of electrical engineering education* 42: 338-349 (2005).
35. Grinder, M.T. A preliminary empirical evaluation of the effectiveness of a finite state automaton animator. *ACM SIGCSE Bulletin*; 35:157-161 (2003).
36. Stallmann, M.F., S.P. Balik, R.D. Rodman, S. Bahram, M.C. Grace. & S.D. High. Proofchecker: an accessible environment for automata theory correctness proofs. *ACM SIGCSE Bulletin* 39: 48-52 (2007).
37. Rodger, S.H. Integrating hands-on work into the formal languages course via tools and programming. In: *International Workshop on Implementing Automata* Ontario, Canada. p. 132-148 (1996).
38. Lee, M.C. An abstract machine simulator. *Lecture Notes in Computer Science* 438: 129-141 (1990).
39. Vieira, L.F., M.A. Vieira. & N.J. Vieira. Language emulator, a helpful toolkit in the learning process of computer theory. *ACM SIGCSE Bulletin* 36: 135-139 (2004).
40. Hamada, M. Supporting materials for active e-learning in computational models. *Computational Science* 5102: 678-686 (2008).
41. McFall, R. & H.L. Dershem. Finite state machine simulation in an introductory lab. *ACM SIGCSE Bulletin* 2: 126-130 (1994).
42. Luce, E. & S.H. Rodger. A visual programming environment for Turing machines. In: *Proceedings of IEEE Symposium on Visual Languages,* Bergen, Norway, p. 231-236 (1993).
43. McDonald, J. Interactive pushdown automata animation. *ACM SIGCSE Bulletin*; 34:376-380 (2002).
44. Grinder, M.T. A preliminary empirical evaluation of the effectiveness of a finite state automaton animator. *ACM SIGCSE Bulletin* 35: 157-161 (2003).
45. Hamada, M. & K.A. Shiina. Classroom experiment for teaching automata. *ACM SIGCSE Bulletin* 36:261-261 (2004).
46. White, T.M. & T.P. Way. jfast: A java finite automata simulator. *ACM SIGCSE Bulletin* 38: 384-388 (2006).
47. Devedzic, V., J. Debenham. & D. Popovic. Teaching formal languages by an intelligent tutoring system. *Educational Technology & Society* 3: 36-49 (2000).
48. Wermelinger, M. & A.M. Dias. A prolog toolkit for formal languages and automata. *ACM SIGCSE Bulletin* 37: 330-334 (2005).
49. Schreyer, B., W. Wawrzynski. Finite automata models for CS problem with binary semaphore. *ACM SIGCSE Bulletin* 38: 330-330 (2006).
50. Meinke K. & N. Walkinshaw. Model-Based Testing and Model Inference. In: Margaria T., Steffen B. (eds) *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change, Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg 7609 (2012).
51. Clarke E. M, O. Grumberg, M. Minea. & D. Peled.

State space reduction using partial order techniques. *International Journal on Software Tools for Technology Transfer* 2: 279-87 (1999).

52. Sipser, M. *Introduction to the Theory of Computation*, 2nd ed. Boston. Thomson Course Technology Boston, USA (2006).
53. Cohen, D.I. *Introduction to computer theory*. 2nd ed. Wiley, New York, USA (1991).
54. Hopcroft, J.E., R. Motwani. & J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation,* 2nd ed. Addison Wesley, New York, USA (2001).
55. Martin, J.C. *Introduction to Languages and the Theory of Computation,* 4th ed. McGraw-Hill, New York, USA (1991).