Teaching-learning methodology for Formal Languages and Automata Theory

Gabriel Spadon de Souza, Celso Olivete Júnior,
Ronaldo Celso Messias Correia, Rogério Eduardo Garcia
Departamento de Matemática e Computação
Faculdade de Ciências e Tecnologia
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Presidente Prudente - SP - Brazil
gabriel@spadon.com.br; {olivete, ronaldo, rogerio}@fct.unesp.br

***Abstract*-Formal languages and automata (FLA) theory have fundamental relevance to the base of knowledge in the computer science area, especially focusing on scientific education. Usually presented by a discipline, the teaching-learning process of FLA is characterized by the high level of abstraction, and it is considered difficult due to the complexity of language formalisms. As support for the learning process, tools have been used to simulate language formalisms. However, the simulation is not enough to reinforce the construction of an abstract concept. In this paper, we present an FLA teaching-learning methodology based on the development of simulators as an approach to clarify the formalism for the students. Through developing their simulators, students are exposed to the data structure and algorithms to handle the formalism. Consequently, students have the opportunity to make the concept concrete.**

Keywords - Methodology, Formal Language and Automata Theory, Learning Tool; Visualization; Cognitive Load; Education; Computing.

I. Introduction

Formal languages and automata theory have fundamental relevance in the computer science area. To a graduate student, especially those who have the academical career as a life goal, it is considered an important knowledge acquired. It is usually presented in a course where the high-level of abstraction characterizes the teaching-learning process of it, and it is seen as difficult due to the complexity of language formalisms.

A common way to exemplify the formalism is using sketches to make a visual representation of an abstract concept. Besides that, drawing take too much time and make usage of static images do not help with the interaction between student and formalisms. For this reason, many tools were used to support the teaching-learning process, and all of them are able to generate a visual illustration of a formalism.

Additionally the use of simulation tools generates an interactive form from an abstraction and helps students to understand the logic behind the visual form. On the other hand, the simulation is not enough to reinforce how the construction of a formalism is done. The teaching-learning process needs to be helped by software capable of formalism interaction, enhancing the learning logic; it still needs to be capable to demonstrate how to create a formalism with programming languages and paradigms.

Several tools have been developed to present formalisms structure [1], [2], [3], [4], [6], [5], [7]. They help the learning process contributing to build a concrete knowledge, minimizing difficulties faced by students with the high level of abstraction [8]. The tools by themselves are not enough, the different perspectives of teaching and developing require methodological approaches to organize the usage of mechanisms to support the teaching-learning process [9].

The experience of modeling a simulator provides the opportunity to the student write their understanding of a formalism in a code architecture [9]. As a result, providing challenges and making connections to different representations of a common abstraction as a Finite Automata and Turing Machine, as many others that can be developed into a common structure.

The main goal of this paper is present a formal language and automata theory teaching-learning methodology based on constructionism [10]. In our methodology, students are exposed to data structure and algorithms, developing their simulators as an approach to clarify the formalism in favor of knowledge construction. Consequently, students have the opportunity to make the concept concrete.

As a result, different years, before and after the methodology application, students' grades were collected and compared yearly. The positive results are observed in students' grades, besides their increasing interest and motivation in learning.

In order to present this teaching-learning methodology, this paper is organized as follows: Section 2 presents some tools available in the literature; Section 3 presents the methodology proposed, and a brief description of a developed simulator, a tool to represent and visualize formalisms and their operations; Sections 4 presents the results of the methodology application; Section 5 presents the final remarks.

II. Related Works

This area covers many classes of formal languages like presented by Chomsky hierarchy [11]. It can be divided into topics where each topic explains and present a formalism with the necessary theory and implementation guidelines.

978-1-4799-8454-1/15/$31.00 ©2015 IEEE

The formalism is related to language, grammars, and regular expression recognition with finite automata. Apart from Turing, Mealy, Moore, Norma, and Post machine; besides the automata variance with determinism, non-determinism, and push-down automata [12].

Our course introduces this area to students, from raw to complex, teaching about:

1) Finite state automata and regular grammars;
2) Push-down automata and context-free grammars;
3) Linear-bounded machine and context-sensitive grammars;
4) Turing, among other machines, and phrase structure;

We use, as a guide, the course topics to compare different formalism simulators. Analyzing each one with their conceptual-implementation. As conceptual understanding, students need to be taught about the theory to represent and solve problems with machines and grammars. In the other hand, the implementation understanding represents the knowledge to create a data structure capable of formalisms recognition [9].

jFAST [1] is a Java finite automata simulator focused on visualization and interactivity to active learning techniques to improve mastery of difficult concepts. It is characterized an easy-to-use graphical software tool for teachers and students, with an emphasis on the introductory level of finite state machine topics. The simulator goal is enhancing teaching effectiveness in this subject, particularly for less advanced computer science students, on the other side, the simulator is limited for automaton representation.

JFLAP [2] is a theoretical computer science framework written initially in C/C++ and after that ported to Java. It was built to design and simulate several variations of finite automata, push-down automaton, one-tape Turing machines and multi-tape Turing machines. Besides that, **JFLAP** allows the conversion between grammars and regular expressions if they are equivalent by Chomsky hierarchy [11]. **JFLAP** is one of the most complete tools developed in this area, it is capable of several conversions from one representation to another.

In the following we present some of the supported conversions [13], [14].

1) nondeterministic finite automaton to a deterministic finite automaton;
2) nondeterministic finite automaton to a regular expression;
3) nondeterministic finite automaton to regular grammar;
4) regular grammar to an automaton;
5) nondeterministic pushdown automaton to a context-free grammar;

Visual **A**utomata **S**imulator (**VAS**) [3] is a tool for simulating, visualizing and transforming finite state automata and Turing Machines. It is characterized to be an application capable of creating and simulate any deterministic or non-deterministic finite automata, as well as Turing Machines. The tool present a complete implementation about Turing Machines and finite automata, but could be improved if others representations were implemented to be used by advanced computer science students.

Auger [4] and **FSM** [5] are simulators that can visually represent a finite automata to grammar recognition, both of them are limited by the finite state automata and regular grammars which represent the lowest level of complexity on Chomsky hierarchy [11]. **FSM** stands out compared to **Auger** because it is an on-line application.

Automatograph [6] is a deterministic finite automata simulator and is a beta software, still in development, which have been used to help in the teaching-learning process with regular grammars and finite state automata. The goal of the application is to validate characters chains with an automaton created by the user and showing the steps of recognition until the automaton reach the final state.

Deus **E**x **M**achina (**DEM**) is a simulator capable to provide a generic platform for designing and running different kinds of automata, such as finite state, push-down, linear bounded automata and Turing, registers and vectors machines, apart from, Markov algorithms. It was built on an icon-based interface where students can sketch an automaton using vertex and edges. The software implements step-by-step run, showing how a given input string is processed as the execution of the automaton. Besides that **DEM** includes save, load and print functions for all representation [13].

A cellular automata simulator [7] is described as a compiler that have been developed for high-speed simulation. Cellular automata are a way to understand and simulate the behavior of complex systems; this simulator is an example of a particular usage of automata to help to understand the usage in a case studying. It is characterized as a temporal and discrete spatial system, capable of updating many cells at high speed with high precision.

As presented there are many tools capable to represent the formal languages and automata theory formalism, and usually the authors present the tools focusing on a feature available. They spent much time creating an icon-based interface to interact with the student, showing how the formalism work, but do not present a way to build it. Also, tools are not bound to a methodology [9]. It is important to use a tool like a support to the teaching-learning process, but an adequate methodological approach is needed to improve the teaching and the learning part, providing a better understanding of the subject.

III. Methodological Approach Proposed: Simulators Construction

Making an analogy to the learning process of data structure and programming language, the capability of understanding an abstraction or a formalism are the most difficulty faced by the students [16]. On the other hand, the highest level of abstraction it is correlated with the programming concept and their theoretical relations [17]. Comparing this points with the formalism faced in formal language and automata theory, we discover that the abstraction, as a step to conquer knowledge, need to be helped by educational software or simulations tools, in the order to simplify and create a visual representation of an abstraction concept. The methodological teaching approach underlies the teaching-learning process and is responsible to guide the knowledge providing strong personal development.

In the following we present a Simulator to illustrate and explain the proposed methodology appliance. The purpose of

this section is to demonstrate, through a simulator developed by a student, how specific and deep knowledge was acquired in the classroom. Focused on finite state automata, regular grammars, Turing Machine and phrase structure.

The stated Simulator is an application based on Multi-Formalism Modeling, and it was developed by a student in JavaFX [15], known as a multimedia framework developed by Oracle Corporation based on multi-platform and web applications development. The current version of the framework allows desktop, browser and mobile development, and it is compatible with Smart TVs, video games, and Blu-rays players. Besides that, JavaFX can be executed by the common JRE or JavaME.

The simulator has a visual icon-based interface to interact with the user. The software front and back-end work together to create, represent and simulate formalisms. The first screen of the simulator is presented in Figure 1, where the user can choose to simulate finite automata or multi-tape Turing Machine.

The tool is capable of working with deterministic and non-deterministic finite automata, regular grammar and expressions, beyond single tape or multi-tape Turing Machines. Apart from the main functions, the software is capable of step-by-step solution, multi-test, regular grammars test and conversion from regular grammar to finite automata, and vice-versa.

Additionally, the student who developed the simulator created a history of tested inputs and a random automaton generator. Besides that, the software allows save images and XML files that can be opened on JFLAP [2].

At next, we present two examples to illustrate the usage of the simulator. The first one is an example of a multiple input test and the second one is a conversion from finite automaton to regular grammar using a random automaton. The result is depicted in Figure 2 and Figure 3 respectively.

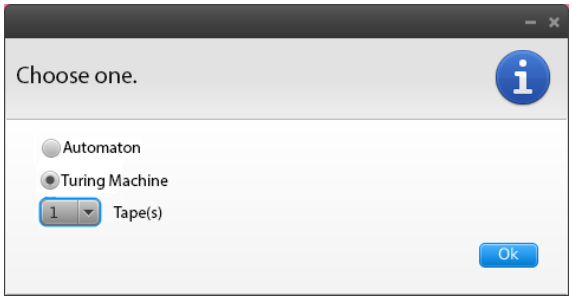

Fig. 1. Start screen of the simulator

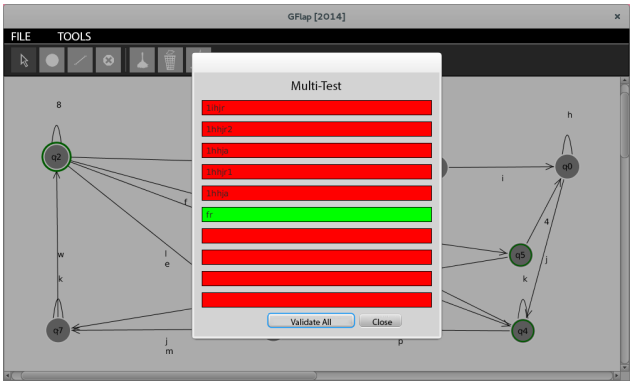

Fig. 2. Testing multiple input

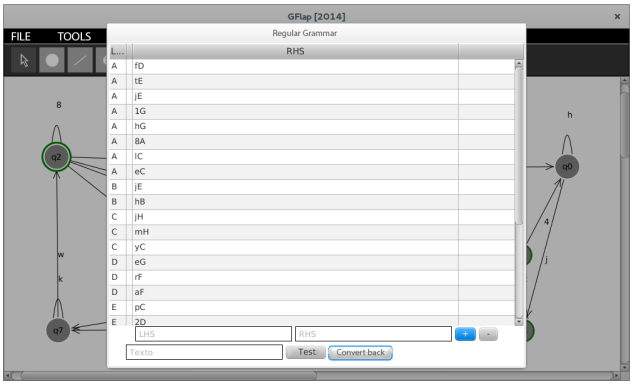

Fig. 3. Linear grammar from finite automaton

By testing random strings, in Figure 2, can be observed red and green field that represents the acceptance of the string input. Besides that, in Figure 3, we can define our own left or right-hand sided grammar and then create and test different inputs on an automaton.

The regular expression and regular grammar checkers are shown in data table where new data addition automatically re-validated the input. If this functionality we can observe in real time the changes to reach a correct string input or a correct grammar/expression declaration. While finite automata and Turing machine works with a start and final state, which can be tested by iterations, working on the automaton or reading and writing a Turing tape to formulate a solution to some problem.

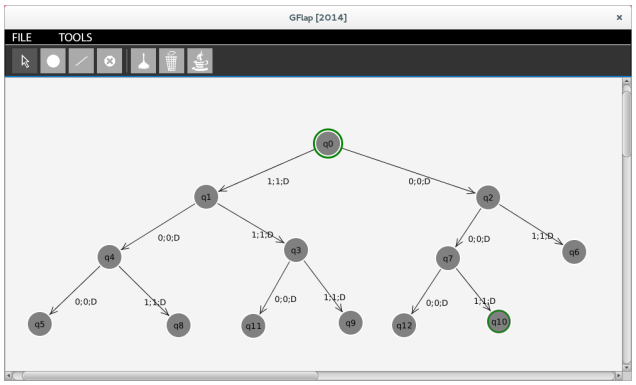

Fig. 4. Turing Machine example

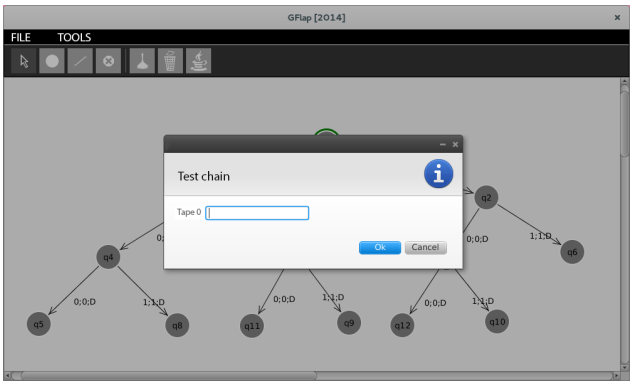

Fig. 5. One tape input for turing machine

The Turing Machine, single and multi-tape function, Figure 4 and Figure 5, are currently in development by the student. The final software will contain the formalisms from two complementary subjects, Formal Language and Automata Theory also to Theory of Computing. The result will be used to encourage the new students on the development, on the way to understand the formalisms in an easier and more didactic approach.

For a final example of usage, we present, in Figure 6 to Figure 13, a full string recognition on a non-deterministic automaton previously presented as a random automaton. The string tested is "1ihj2r". On each step of recognition, if the single character is accepted, the color will be changed to blue, else the recognition will stop. The same rule is applied in each state if automaton state amend the last state will turn green, else stay gray and the recognition process stop.

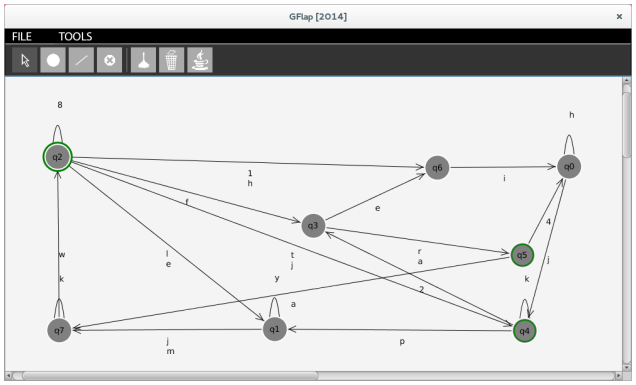

Fig. 6. Deterministic finite automaton

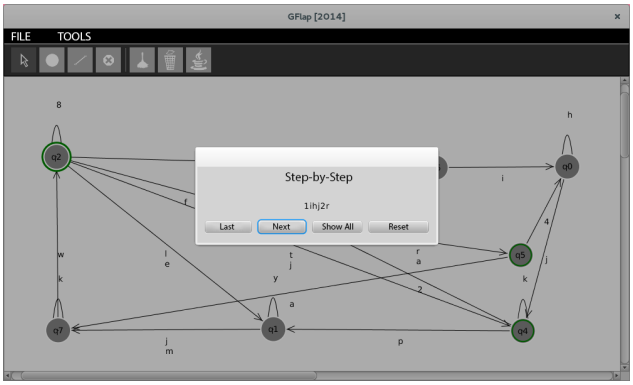

Fig. 7. Step-by-step run on random input

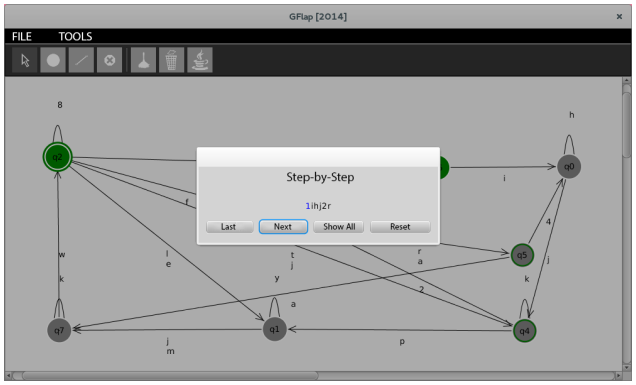

Fig. 8. Step 1, accept "1"

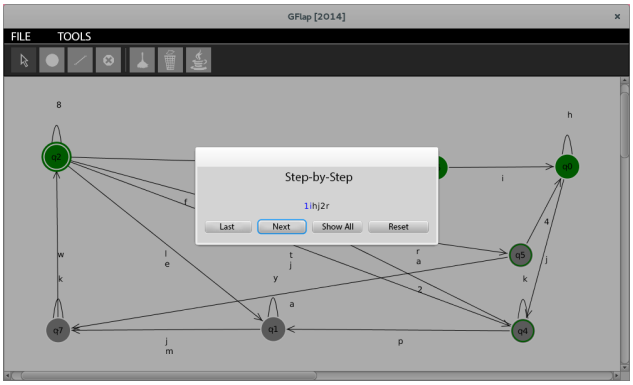

Fig. 9. Step 2, accept "i"

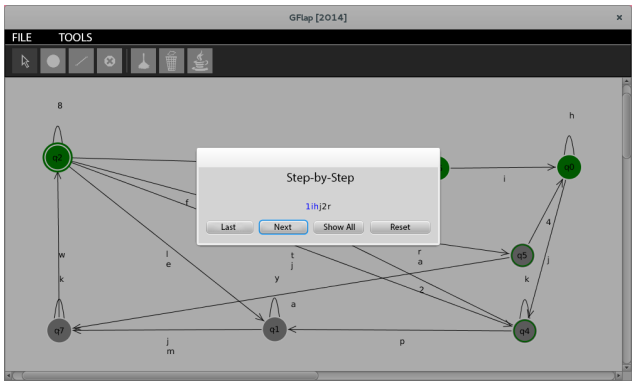

Fig. 10. Step 3, accept ”h”

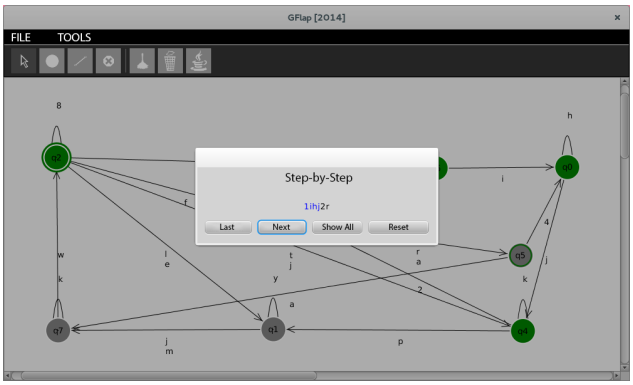

Fig. 11. Step 4, accept ”j”

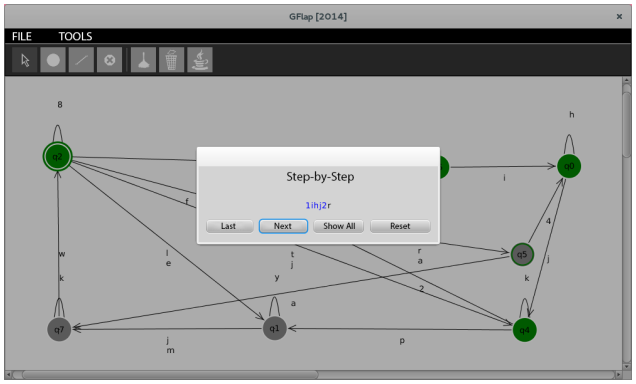

Fig. 12. Step 5, accept ”2”

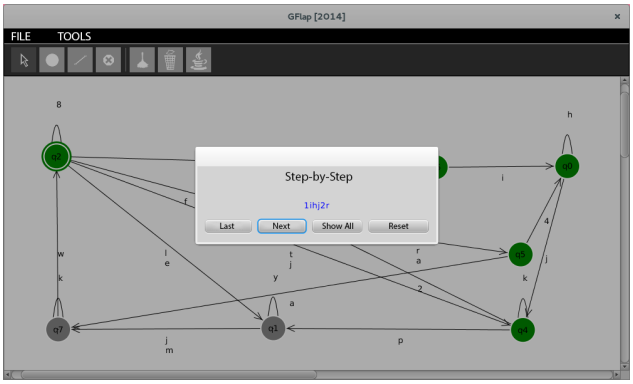

Fig. 13. Step 6, accept ”r”, accept character chain

IV. Appliance Results

The ease of understanding the abstraction and the formalism construction are possible by interaction with the finite automaton and the regular languages. The icon-based interface enables the interaction with the tool and allows visual operations, structure changes, and real-time viewing results. As a result of simulator development, different years, before and after the methodology appliance, students' grades were collected and compared yearly, and they are depicted in Figure 14 and Figure 15.

These results support the teaching-learning methodology, once it is fundamental for students in the computer science area, and it involves formalism recognition and languages generation that are used in compilers and programming languages. The positive results are observed in students' grades. We separate the results in year grades (Figure 14) and simulator grades (Figure 15).

At Figure 14, is presented an over years time-line, considering classes from 2012 to 2014, presenting students' grades. We can verify that the grades have become, on average, higher, compared to previous years. The final grade for each student presented above is an important factor to understand the results because the formula helps to see why a student has a bad or good grade at the end of the subject.

The minimum grade to approve a student is 05.00 of 10.00 points. To reach the points students are evaluated with one theoretical test on each trimester and the simulator presentation at the end of the semester. The student grade is the average of all evaluations made during the semester, but we use a clause to make the average or not. First of all, we check the grades, and if both grades, tests, and simulator, are greater than or equal to 05.00, we sum grades and make an average. Else, the final grade is the lowest grade between test and simulator. The formula is presented at next.

$$test(T1, T2) = \frac{T1 + T2}{2};$$

$$final(T1, T2, SIM) = \frac{test(T1, T2) + SIM}{2};$$

$$grade(T1, T2, SIM) =$$

$$\begin{cases} final(T1, T2, SIM), & \text{if } test(T1, T2) \geq 5.0 \\ & SIM \geq 5.0; \\ lowest(test(T1, T2), SIM), & \text{otherwise;} \end{cases}$$

Considering the formula, the results shown in Figure 14 does not focus its values on A-concept (grades between 8.0 and 10.0), the trend line denotes that the values are higher in its means, which shows that over the years more students are approved. The higher number of approved students is a result of the teaching-learning methodology, which made them more focused and stimulated in classes.

Besides the main results, we can observe an increase of students with grades between 3.0 and 4.0 in the years 2013 and 2014. Each case has examined separately by the teacher,

and it was concluded that these students, in particular, made an incomplete development of the proposed simulator. Despite the fact that the student has achieved good test scores, the final grades clause give to that students the lowest grade, in this case, the simulator grade.

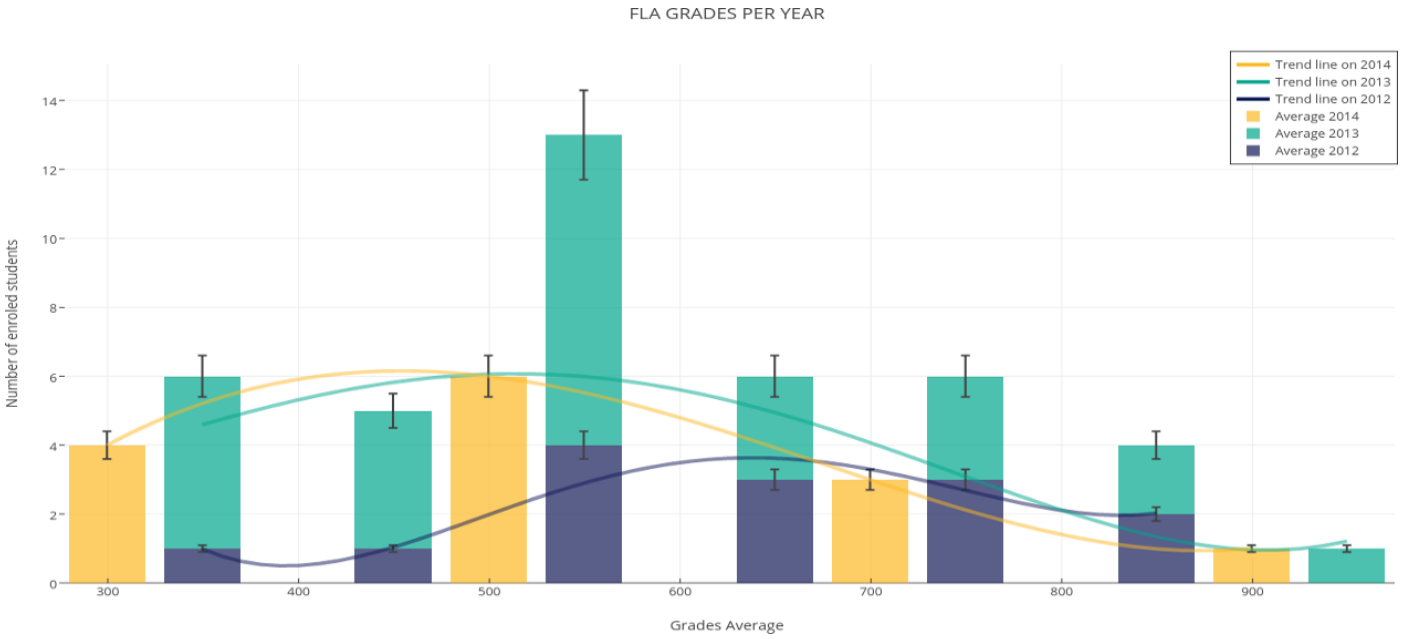

Fig. 14. Application based on Multi-Formalism Modeling

In Figure 15, is depicted the students simulator grades. The values follow a distribution that results in an arc on year 2014, while for 2012 and 2013, results are close to a sinusoidal line with growth between grades 3.0 and 5.0, which features a lower average between these years and less favorable values.

In both cases, Figures 14 and 15, we emphasize that the number of students between the years is not constant, which changes the size of graphics and the position of the trend lines. However, to make any analyze we just considered the percentage of median results.

Having the discussed statistical analysis and the results depicted in the graph in Figure 14 and 15, it proves that the teaching-learning methodology is crucial to knowledge construction process.

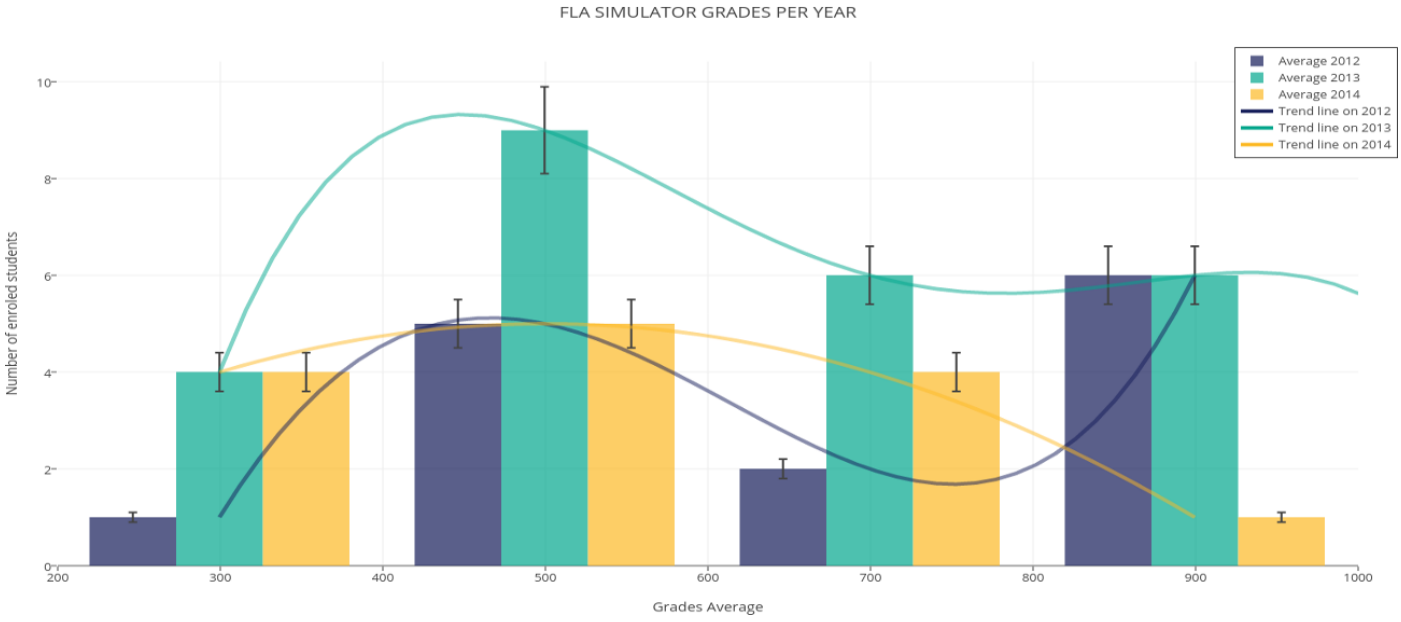

Fig. 15. Application based on Multi-Formalism Modeling

V. Conclusion

In this paper, we present a teaching-learning methodology to support the formalism model construction about formal languages and automata theory. According to the methodology proposed, for each topic about the subject, the teacher must propose a simulator development as classwork or homework to guide the student on formalism understanding.

In order to present our methodology, we use a developed simulator, based on Multi-Formalism Modeling and it was developed by a student in JavaFX [15]. The simulator had a visual icon-based interface to interact with the user and was built to support students understand of FLA formalisms. The visual resources and the programming logic to built it aids the student to learn abstract concepts easier, in comparison to imagining how formalisms work and how they are represented.

This methodology was applied at São Paulo State University (UNESP), applying the methodology proposed during four years, considering the current year, 2015. Our evaluation consisted in compare students' grades yearly, before and after, the methodology appliance.

The ease of understanding and building the abstraction represented by the formalism are possible by the interaction with the formalism. User interaction is possible by defining operations, changes in the structure and real-time viewing results. It is intended to follow this proposed teaching methodology with the next class of LFA and extend the teaching-learning methodology to Theory of Computing, aiming to incorporate formalities related to context-free languages, context-sensitive languages, and recursively enumerable languages.

In general, the positive results are observed in students' grades and their evaluation. Also, according to the teacher, the students have shown interest and more motivation.

Acknowledgment

We are grateful to CNPq – "National Counsel of Technological and Scientific Development" by the support – Process 457875/2014-3, and to the Department of Mathematics and Computer Science (DMC) at "Faculdade de Ciências e Tecnologia, UNESP – Universidade Estadual Paulista" by the resources granted to this research.

References

[1] White, T.M.; Way, T., "jFAST: a java finite automata simulator". Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education, SIGCSE 2006, Houston, Texas, USA, March 3-5, 2006 doi: 10.1145/1124706.1121460

[2] Rodger, S.H.; Qin, H.; Su, J., 2011. "Changes to JFLAP to increase its use in courses". In Proceedings of the 16th annual joint conference on Innovation and technology in computer science education (ITiCSE '11). ACM, New York, NY, USA, 339-339. DOI=10.1145/1999747.1999851 http://doi.acm.org/10.1145/1999747.1999851

[3] Bovet, J., Visual "Automata Simulator" 2006 [website]. Available: https://www.cs.usfca.edu/ jbovet/vas.html

[4] Evoluma. "Auger - Ambiente para construção e simulação de autômatos finitos" [website]. Available: http://www.evoluma.com/auger/index.html

[5] Zuzak, I.; Jankovic, V., "FSM simulator" [website]. Available: http://ivanzuzak.info/noam/webapps/fsm_simulator/

[6] Kienetz, E.B.; Canal, A.P., "Deterministic finite automata simulator - Automatograph". Disc. Scientia, Ciências Naturais e Tecnológicas, S. Maria, v. 5 , n. 1, p. 1-9, 2004 issn: 1519-0625

[7] Akamine, Y.; Endo, S.; Yamada, K., "The development of a compiler for cellular automata simulator", Systems, Man and Cybernetics, 2003. IEEE International Conference on , vol.4, no., pp.3887,3892 vol.4, 5-8 Oct. 2003 doi: 10.1109/ICSMC.2003.1244495

[8] Ferreira, B.J.P., "Em Busca de uma Prática Pedagógica Inclusiva: Uma Experiência no Ensino de Estruturas de Dados Mediada por Animações Gráficas". XV Workshop sobre Educação em Informática. Anais do XXVII Congresso da SBC. Rio de Janeiro, 2007.

[9] Messias Correia, R.C.; Garcia, R.E.; Olivete, C.; Costacurta Brandi, A.; Cardim, G.P., "A methodological approach to use technological support on teaching and learning data structures," Frontiers in Education Conference (FIE), 2014 IEEE, pp.1,8, 22-25 Oct. 2014 doi: 10.1109/FIE.2014.7043992

[10] PAPERT, S.; I. HAREL.,"Situating Constructionism". Constructionism, Ablex Publishing Corporation. 1991.

[11] Chomsky, A.N., 1956. Three models for the description of language. Information Theory, IRE Transactions on, v.2(3), pp. 113–124.

[12] Hopcroft J.E.; Ullman J.D., 1990. "Introduction to Automata Theory, Languages, and Computation" (1st ed.). Addison-Wesley Longman Publishing, Boston, MA, USA.

[13] Chesñevar, C.I.; Cobo, M.L.; Yurcik, W., 2003. "Using theoretical computer simulators for formal languages and automata theory". SIGCSE Bull. 35, 2 (June 2003), 33-37. DOI=10.1145/782941.782975 http://doi.acm.org/10.1145/782941.782975

[14] Chesñevar, C.I.; González M.P.; Maguitman, A.G., 2004. "Didactic strategies for promoting significant learning in formal languages and automata theory". In Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education (ITiCSE '04). ACM, New York, NY, USA, 7-11. DOI=10.1145/1007996.1008002 http://doi.acm.org/10.1145/1007996.1008002

[15] Field, R., "JavaFX Language Reference - Sun Microsystems" [website]. Available: ¡http://openjfx.java.sun.com/current-build/doc/reference/JavaFXReference.html¿

[16] Ribeiro, R.S.; Brandão, L.O.; Brandão, A.A.F., 2012. "Uma visão do cenário nacional do ensino de algoritmos e programação: uma proposta baseada no paradigma de programação visual", I Congresso de brasileiro de Informática na educação, 2012, Rio de Janeiro, RJ - Brasil.

[17] Souza, D.M.; Maldonado, J.C.; Barbosa, E.F., 2012. "Aspectos de Desenvolvimento e Evolução de um Ambiente de Apoio ao Ensino de Programação e Teste de Software". I Congresso de brasileiro de Informática na educação, 2012, Rio de Janeiro, RJ - Brasil.