

Language Emulator, a Helpful Toolkit in the Learning Process of Computer Theory

Luiz Filipe M. Vieira, Marcos Augusto M. Vieira, Newton J. Vieira
Department of Computer Science
Federal University of Minas Gerais
{lfvieira, mmvieira, nvieira}@dcc.ufmg.br

ABSTRACT

Language Emulator, written in Java, is a toolkit to help undergraduate students to understand the concepts of Automata Theory. The software allows the manipulation of regular expressions, regular grammars, deterministic finite automata, nondeterministic finite automata with and without lambda transitions, and Moore and Mealy machines. Language Emulator introduces error-detecting and internationalization functionalities into automata tools. It has been accepted by 95% of students in a recent survey, indicating that it is a helpful toolkit in learning Automata Theory.

Categories & Subject Descriptors

K.3.2 *Computers and Education*: Computer and Information Science Education Computer science education.

General Terms

Experimentation, Languages, Theory.

Keywords:

Computer Theory, tool, teaching.

1. INTRODUCTION

Computer Theory is traditionally taught without any software assistance, being limited to tasks not requiring computer use. Here we suggest having some problem solving based in the use of a software toolkit.

Rodger [12] has shown that there are two main reasons for the difficulties found in courses concerning Computer Theory. The first reason is that such courses require more prior knowledge in mathematics than other courses in computer science. The second reason is that students do not have a feedback while working with non-interactive problems. The difficulties can be minimized by the use of a toolkit, such as Language Emulator, which allows students to learn Computer Theory interactively.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGCSE'04, March 3–7, 2004, Norfolk, Virginia, USA.
Copyright 2004 ACM 1-58113-798-2/04/0003 ...\$5.00.

In this paper, the related bibliography is described in Section 2, the functionalities of Language Emulator and the graphic interface are presented in Section 3 and Section 4, respectively. Section 5 describes how Language Emulator can be used in a classroom by means of a case study. Finally, Section 6 brings some conclusions and new perspectives.

2. RELATED WORKS

Many courses in Computer Science have toolkits that help students in the learning process. Many tools, including [7], have been developed in order to create animations of algorithms which can be used for teaching.

JFLAP [5][12][13][9][11] (Java Formal Languages and Automata Package) is a package of graphical tools that can be used both in teaching and learning the basic concepts of Automata Theory. It allows designing and running several variations of automata, such as deterministic finite automata, nondeterministic finite automata, pushdown automata and Turing Machines. However, it accepts English language only, and it does not support transition among the several types of automata that it works with [6].

Deus Ex Machine [1], developed by Nick Savoie, comprises simulations of seven computation models. It was designed to be used in parallel with the book “Models of Computation and Formal Languages”. It provides a generic multi-purpose platform for designing and simulating different kinds of automata, including deterministic finite automata, nondeterministic finite automata, pushdown automata and Turing Machines. Nowadays, there are versions available for the Windows operating system only.

Turing’s World [3] is a graphical tool that allows to specify Turing machines by drawing the state diagram and their transitions. The user can visualize, graphically, the operations on the Turing Machine.

Webworks Laboratory [4] presents a set of projects in Computer Theory. Its main highlight is the “Animating the Theory of Computing”, a tool developed in order to allow students to learn with animations of finite automata, interactively.

Ganimal [2] is a project of software for compilers learning that generates animations of abstracts machines. A machine can be chosen according to the type of language: imperative, functional or a stack.

The Java Computability Toolkit (JCT) [10] is a graphical environment to create and simulate deterministic finite automata, nondeterministic finite automata and Turing Machines. It was developed in Java; it allows minimizing a deterministic finite automaton, and to convert a nondeter-

ministic finite automaton into a deterministic finite automaton. As in JFLAP, automata can be drawn on a canvas by positioning states and transitions. Finite automata can be executed on arbitrary input strings and executed in step-by-step fashion with immediate visual feedback [6].

Table 1 summarizes the related works. Int. stands for internationalization, meaning that the software is available in more than one language. Err. is an abbreviation for error-detecting, a feature that allows the verification of a given automaton. Deterministic finite automata (DFA), nondeterministic finite automata (NFA), nondeterministic finite automata with lambda transitions (NFA λ), regular grammars (RG), regular expressions (RE), Mealy machines (Mealy), Moore machines (Moore), pushdown automata (PDA) and Turing machines (TM) are the paramount types of formal languages. Plat. indicates the platforms the tools are available for, and finally, av. means the availability of the software. Each of the related tools has different features. Language Emulator is the only work that supports internationalization and error-detecting functions (as we explain in Section 5).

3. TOOL

The Language Emulator was developed to work with the most usual forms of regular languages. The seven formats supported are:

- Regular Expression (RE).
- Regular Grammar (RG).
- Deterministic Finite Automata (DFA).
- Nondeterministic Finite Automata (NFA).
- Nondeterministic Finite Automata with lambda transitions (NFA λ).
- Moore Machine.
- Mealy Machine.

Additionally, Language Emulator has the following functionalities:

- Minimize DFA.
- It transforms any format, supported by Language Emulator, into an equivalent one.
- It generates the words of size $< n$ that belongs to a RG.
- It determines whether a given word belongs to the language of a RG.
- It determines whether a word belongs to the language correspondent to a DFA.
- It determines whether a word belongs to the language correspondent to a RE.
- It transforms a Mealy machine into a Moore machine.
- It transforms a Moore machine into a Mealy machine.
- It determines the output of a Moore machine for a given input.

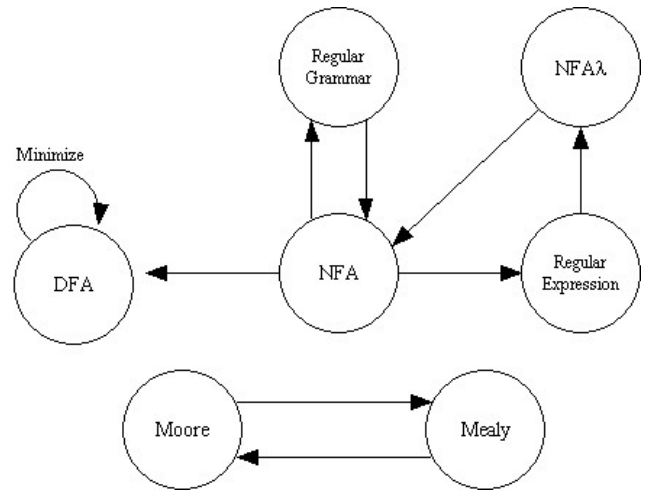


Figure 1: Transformation among the formats of regular language.

- It determines whether a word belongs to a Mealy or Moore machines.
- Union, intersection and difference of two DFAs.
- Complement of a DFA.
- Union of the differences of two DFAs. $A_F = (A_1 - A_2) \cup (A_2 - A_1)$
- Save and Load a DFA.
- Export a DFA to a file in the format accepted by the GraphViz [8] tool. GraphViz receives a textual description of a graph and generates its visual representation.

Figure 1 shows the transformations supported by the Language Emulator. The toolkit also shows the intermediary steps of algorithms, printing the values at each phase of execution.

4. INTERFACE

Java was the programming language chosen to create the toolkit for Automata Theory because it works in any platform supporting Java Virtual Machine which is common in educational environments. The interface was designed using Java Swing, a standard in the Java platform. Initially, the toolkit was created in Portuguese and English, but it is possible to internationalize it, switching to the desired language as, for example, Spanish.

There are seven tabs the user can chose to construct any of the seven formats supported by Language Emulator: DFA, NFA, NFA λ , RG, RE, Moore and Mealy machines. For each format there is an appropriate interface. This can be visualized when the user selects the tab corresponding to the format's name. It allows the user to access the functionalities presented in the format. For example, the DFAs can be minimized or generate the regular expression that recognizes the same language as DFA does. These two functions can be accessed by clicking the buttons Minimize and RE presented in the DFA tab (Figure 2).

Software	Int.	Err.	DFA	NFA	NFA λ	RG	RE	Mealy	Moore	PDA	TM	Plat.	Av.
Language E.	✓	✓	✓	✓	✓	✓	✓	✓	✓			all	free
JFLAP			✓	✓	✓	✓				✓	✓	all	free
Deus Ex-M.			✓	✓	✓					✓		windows	free
Turing's W.											✓	macintosh	com.
Webworks			✓	✓	✓	✓						all	free
Ganimal			✓	✓	✓							all	free
JCT			✓	✓	✓						✓	all	free

Table 1: Comparative Table.

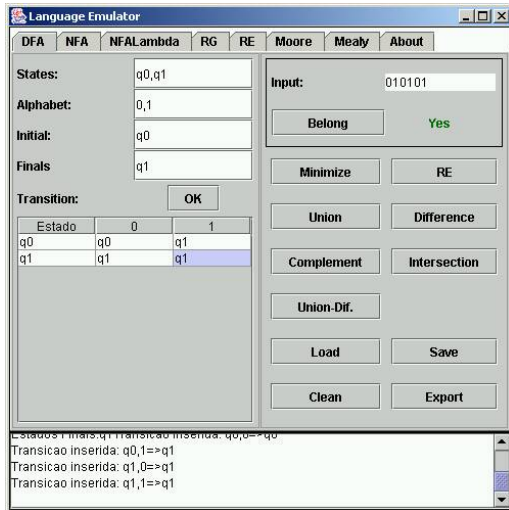


Figure 2: Deterministic Finite Automaton at Language Emulator.

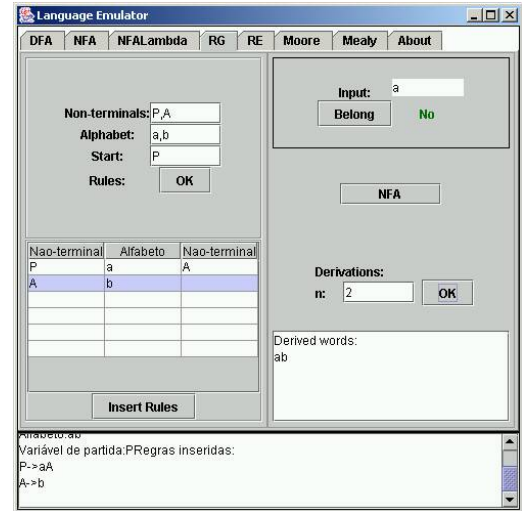


Figure 4: A Regular Grammar in Language Emulator

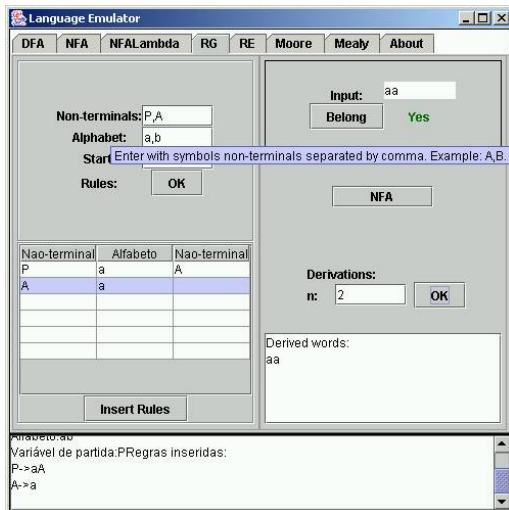


Figure 3: ToolTips as presented in Language Emulator

Figure 3 is a screen shot of Language Emulator showing the presence of ToolTips which make the toolkit easier to be used.

Figure 4 shows the construction of a regular grammar. Initially, the user introduces the nonterminals, the alphabet and the initial nonterminal according to the definition of a regular grammar. The user starts to fill the table where the rules are inserted by clicking the button OK. The grammar is built by clicking the button Insert Rules. It is possible to check whether a given word belongs to the regular language generated by the RG or to construct the equivalent NFA.

With the NFA, it is possible to find whether a word is recognized by the automaton, to modify it, to insert new symbols in its alphabet and to change its initials and/or finals states. It is also possible to generate an equivalent regular expression, grammar or DFA.

It is possible to modify any components (states, alphabet, initial state, final state and transitions) of any automaton. The DFA can be minimized or the equivalent regular expression can be generated.

The Language Emulator can be used to verify whether a word belongs to a language denoted by a regular expression and/or generate the equivalent NFA. Both can be edited by the user. In the Language Emulator, representations in the format of a NFA λ and in the DFA format have similar functionalities. It is possible to check whether a word is recognized by the automaton, to modify it and to generate the equivalent NFA.

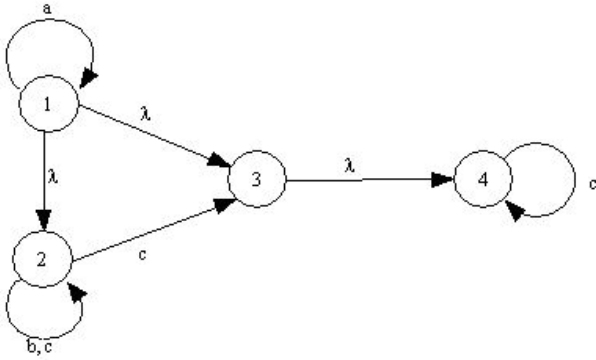


Figure 5: Automata's Exercise

5. TEACHING

In parallel to the theory, some practical exercises can be assigned. As each part of the theory is taught, a task could be assigned using the Language Emulator.

5.1 Case Study - UFMG

Language Emulator was first used in the course “Fundamentals of the Theory of Computation” offered by the Dept. of Computer Science at Federal University of Minas Gerais (first semester, 2002). After NFA λ , DFA and RE had been taught, the following exercise was proposed:

Example of exercise

Considering the NFA λ in Figure 5:

- Construct, using paper, the transition table.
- Construct, using the toolkit, the NFA λ .
- Construct, using paper, the equivalent NFA.
- Using the toolkit, convert the NFA λ into a NFA.
- Construct, using paper, the equivalent DFA.
- Using the toolkit, convert the NFA into a DFA.
- Minimize, using paper, the DFA.
- Using the toolkit, minimize the DFA.
- Construct, using paper, the regular expression that denotes the language recognize by the automata.
- Using the toolkit, convert the DFA into a RE.

The exercise was applied as homework, so the students could correct their own exercises noticing where their mistakes were.

5.2 Helping the teacher and students

The toolkit may be useful for the teacher. It allows the computation of a resulting regular language from union, intersection or difference of two regular languages. Furthermore, the operation *union of differences*¹ of two regular languages was embedded into the toolkit. This operation is very useful, as we show next.

¹This operation is also known as *symmetric difference*.

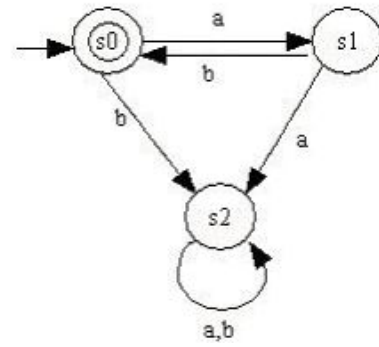


Figure 6: Correct DFA that recognize L_2

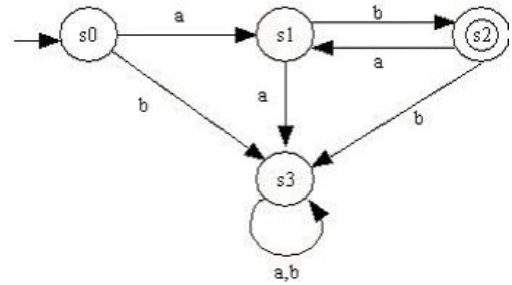


Figure 7: Student's DFA

Let L_1 be the language recognized by automaton A_1 described by a student, and let L_2 be the language recognized by the correct automaton A_2 , then selecting the *union of differences* button in the tab of DFAs, an automaton A_3 is generated that accepts the language $(L_1 - L_2) \cup (L_2 - L_1)$. This language is empty if and only if $L_1 = L_2$, meaning in this case that A_1 is correct. It is also possible to show the students examples of words that cause the automaton A_1 to behave incorrectly: words of the language accepted by A_3 .

For example, let the language L_2 be equal to $\{ab\}^*$. Figure 6 shows a correct automaton A_2 for this language. Figure 7 shows an automaton A_1 constructed by a student. The automaton A_1 recognizes $\{ab\}^+$, instead of $\{ab\}^*$. This mistake is easily shown by means of a counter-example generated by calculating the difference $L_2 - L_1 = \{\lambda\}$. In this example, it clear that A_1 does not recognize λ and, therefore, does not recognize the language L_2 . In general, by using the *difference* button from the toolkit it is possible to find the set of words $L_2 - L_1$ that should be but are not recognized by automaton A_1 .

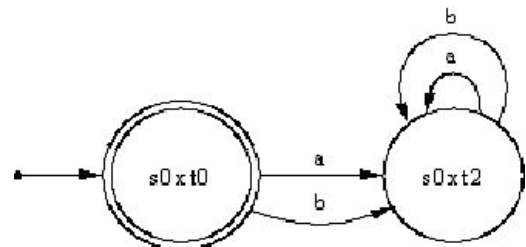


Figure 8: Resulting automaton from $A_2 - A_1$

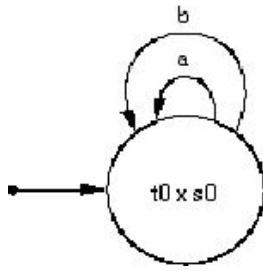


Figure 9: Resulting automaton from $A_1 - A_2$

The automaton in Figure 8 recognizes only λ . We can also generate the automaton that recognizes $L_1 - L_2$ in order to find the words that are incorrectly recognized by automaton A_1 . The automaton in Figure 9 recognizes the empty set. Note that there is no final state. Figures 8 and 9 were generated with GraphViz [8] using the export function from Language Emulator. A second manner to check the correctness of the student's automaton is to generate the equivalent regular expression and verify if it represents the same language that the correct automaton does. In our example, the Language Emulator returned $ER_1 = (ab(ab)^*)$ for the student's automaton. Note that λ does not belong to the language equivalent to ER_1 .

5.3 Toolkit Evaluation

Since it was the first time the Language Emulator was used in a classroom, the students were invited to evaluate the toolkit.

Nearly 95% of the students answered that Language Emulator was helpful in the learning process of Automata Theory. They have also given some suggestions in order to improve the toolkit as well its capability to help in the teaching process. The greatest advantage cited by the students was that they could solve an exercise using paper and pencil and, after that, use the Language Emulator to verify if they did it right.

A better evaluation would consist of a comparison among related work by the students. Unfortunately, it was not possible due to the fact that Language Emulator was the only internationalized toolkit.

6. CONCLUSION

The Language Emulator, which introduces error-detecting and internationalization functionalities into automata tools, has been used in the process of teaching Automata Theory. It is possible for students to interact with the main formalisms related to regular languages and receive an immediate feedback. The toolkit use had a high level of approval, agreeing with the finality of this toolkit. The students liked the fact that they could verify if their work, done previously in paper, was correct.

The toolkit is in an earlier stage and it can be improved. Future works include the drawing of states and transitions, using a graphical input, and the research of new techniques that can make the understanding of the automata theory easier. One such topic of research is to show, using the least number of modifications, how to transform the student's automaton from incorrect to correct.

Acknowledgment

This work was partially supported by CNPq.

7. REFERENCES

- [1] Deus ex machine, <http://www.ics.uci.edu/savoivu/dem/>. Web site.
- [2] Ganimal <http://rw4.cs.uni-sb.de/ganimal/>. Web site.
- [3] Turing's world, <http://www-csli.stanford.edu/hp/logic-software.html>. Web site.
- [4] Webworks laboratory, <http://www.cs.montana.edu/webworks>. Web site.
- [5] Bilska, A. A collection of tools for making automata theory and formal languages come alive. *SIGCSEB: SIGCSE Bulletin* 29 (1997), 15–19.
- [6] Chesnevar, C. I., Cobo, M. L., and Yurcik, W. Using theoretical computer simulators for formal languages and automata theory. *SIGCSEB: SIGCSE Bulletin* 35 (2003), 33–37.
- [7] Gloor, P. Aace - algorithm animation for computer science education. In *Workshop on Visual Languages* (1992), pp. 25–31.
- [8] Koutsoufios, E., and North, S. C. Drawing graphs with dot. In *"AT&T Bell Laboratories"* (Murray Hill NJ, U.S.A., October 1993).
- [9] Procopiuc, M., Procopiuc, O., and Rodger, S. H. Visualization and interaction in the computer science formal languages course with jflap. *ASEE/IEEE Frontiers in Education (FIE) Conference* (1996), 121–125.
- [10] Robinson, M., Hamshar, J., Novillo, J., and Duchowski, A. A java-based tool for reasoning about models of computation through simulating finite automata and turing machines. In *30th Annual ACM SIGCSE Symposium* (New Orleans, Louisiana, March 1999).
- [11] Rodger, S. H. Integrating hands-on work into the formal languages course via tools and programming. *Workshop on Implementing Automata, Lecture Notes In Computer Science* 1260 (1996), 132–148.
- [12] Rodger, S. H., and Gramond, E. Using JFLAP to interact with theorems in automata theory. *SIGCSEB: SIGCSE Bulletin* 31 (1999), 336–340.
- [13] Rodger, S. H., and Hung, T. Increasing visualization and interaction in the automata theory course. *SIGCSEB: SIGCSE Bulletin* 32 (2000), 6–10.