Simulators for Teaching Formal Languages and Automata Theory: A comparative Survey

Carlos Iván Chesñevar **María Laura Cobo**

Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Av.Alem 1253 – B8000CPB Bahía Blanca – República Argentina
Tel/Fax: (+54) (291) 459 5135/5136
Email: {cic,lc}@cs.uns.edu.ar – Web: http://cs.uns.edu.ar/∼cic
Key words: Educational Software, Automata Theory, Formal Languages

Abstract

Formal languages and automata theory (FL&AT) are central subjects in the CS curricula which are usually difficult both to teach and to learn. This situation has motivated the development of a number of *computer simulators* as educational tools which allow the student to implement and 'bring to life' many topics which traditionally were studied and analyzed mathematically rather than algorithmically.

This paper discusses the main features of several educational software tools currently available for teaching FL&AT. Advantages and weaknesses of different tools are analyzed and contrasted. Based in our experience, some rationales and practical considerations for the development of this kind of educational tools are proposed.

1 Introduction and motivations

Formal languages and automata theory are central subjects in the CS curricula which are usually difficult both to teach and to learn. On the one hand, the professor faces the challenge of presenting motivating lectures on topics such as formal languages, pushdown automata or Turing machines, which are very abstract compared to any programming language taught in a previous CS1 course. On the other hand, many students find difficult to grasp the importance of these new concepts or even their intuitive underlying ideas as they are overwhelmed by the abstract formal notation being used.

This situation has motivated the development of a number of *computer simulators* as educational tools which allow the student to implement and 'bring to life' many topics which traditionally were studied and analyzed mathematically rather than algorithmically. In other areas of Computer Science (such as in computer architecture [YYP01]), software simulators provide an excellent alternative for combining different complex issues into a single simulation environment (e.g. simulating both micro instructions at chip level and assembly instructions at processor level). In that respect, FL&AT can be seen as a specialized area of simulating computer architectures, [1] where *theoretical computer simulators* are considered.

Theoretical computer simulators have some advantages with respect to their real-world counterparts. On the one hand, they rely on just a few (and rigorously related) theoretical

[1] See [YYP01, YG02] for excellent surveys about the state of the art of computer simulators for computer architectures.

models, whereas a plethora of different architectures for actual computers can be envisioned. On the other hand, these theoretical models are conceptually simple, making easier to strengthen pedagogical issues when designing a simulation software.

There exist many software tools for helping to teach formal languages and their related automata. We can distinguish two significant categories, namely:

1. Generic, multi-purpose software packages for teaching and integrating several related concepts of FL& AT.

2. Software tools oriented towards simulating a specific class of automata with educational purposes.

In this paper we will review the main features of several software tools available for teaching *FL&AT*. Most of these tools have been experimented during the undergraduate course "Fundamentos de Ciencias de la Computación" at the Departamento de Ciencias e Ingeniería de la Computación (Universidad Nacional del Sur, Bahía Blanca, Argentina). Advantages and weaknesses of introducing such software tools are discussed.

The paper is structured as follows. First, in section 2 generic, multi-purpose software tools for modeling formal languages and their related automata are discussed. Then, in section 3 we discuss different software simulators for specific kinds of automata, such as Turing Machines, Moore transducer automata and Petri Nets. In Section 4 we briefly describe our experience in incorporating simulators as a teaching aid during an undergraduate course in FL&AT. We also mention some basic design rationales which in our opinion are useful for guiding the development of such software teaching tools. Finally, section 5 concludes.

2 Multi-purpose software tools for *FL&AT*

Multi-purpose software tools for *FL&AT* aim at integrating different computer simulators into a single, unified package. Two major software tools of this kind have been identified: DEUS EX MACHINA and JFLAP. Next we will describe their main features.

- DEUS EX MACHINA is a software tool developed by Nicolae O. Savoiu and comprises simulations of seven models of computation covered in the textbook *"Formal Models of Computation"* [Tay98]. It provides a generic multi-purpose platform for designing and running different kinds of automata, such as finite state automata (FSAs), push-down automata (PDAs), Turing Machines (TMs), register machines (RMs), vector machines (VM), linear-bounded automata (LBAs) and Markov algorithms (MAs).

  Each of these automata can be defined using a natural icon-based interface. Students can define an automaton by drawing it using nodes and arcs. Automata can then be run in a step-by-step fashion, showing how a given input string is processed as the execution of the automaton proceeds. DEUS EX MACHINA includes facilities for students to save, load and print their own automata. The software is freely distributed, and it can be found at `http://www.ici.uci.edu/~savoiu/dem`.

- JFLAP is a package of graphical tools can be used as an aid in learning the basic concepts of Formal Languages and Automata Theory [GR99]. The original version

(FLAP) was written in C/C++ for X-window based systems; JFLAP is the corresponding Java version. JFLAP was designed by Susan Rodger[2] and her research group at Duke University, and it allows to design and simulate several variations of finite automata (FA), pushdown automata (PDA), one-tape Turing machines (TM) and two-tape Turing machines (TTM).

Additionally, JFLAP allows to input grammars(GRM) and regular expressions(REX) and convert them between each other. Features of JFLAP include several conversions from one representation to another. The conversions are nondeterministic finite automaton (NFA) to deterministic finite automaton (DFA), DFA to minimum state DFA, NFA to regular expression, NFA to regular grammar, regular grammar to NFA, nondeterministic pushdown automaton (NPDA) to context-free grammar (CFG), and three algorithms for CFG to NPDA.

The software is freely distributed, and it can be found at `http://www.cs.duke.edu/~rodger/tools/`.

2.1 Contrasting Deus Ex Machina and JFLAP

As in most *FL&AT* courses, students have first to deal with finite state automata (FSA), and then with pushdown automata (PDA). During the undergraduate course "Fundamentos de Ciencias de la Computación" we have encouraged students to use different computer simulators for designing and running their automata. Deus Ex Machina and JFLAP were two packages that were shown and used in lectures for complementing theoretical concepts and 'running' FSAs and PDAs.

According to our experiences, Deus Ex Machina was a good choice to start introducing the use of a software package as a tool for solving typical automata exercises. Deus Ex Machina encourages students to understand the importance of a formalism as they necessary have to provide the alphabet to be used, the corresponding input and tape language, etc. *before* they can run a given automaton. In contrast, JFLAP is more intuitive, not enforcing so many formal features. We combined the use of both packages to get 'the best of both worlds'. First we introduced Deus Ex Machina, in order to emphasize the importance of notions such as input language, tape language, etc. (which seem a nuisance for many students when designing automata). Later, when students were more familiarized with notation issues, we switched into JFLAP.

It must be noted that both Deus Ex Machina and JFLAP provide an interactive design screen, in which students can 'draw' an automata by using appropriate tools (such as 'make new state', 'make new arc', etc.) from a toolbar. In Deus Ex Machina nodes and arcs can be enriched by additional comments. However, drawings are not fully resizable (nodes and fonts are always the same size), which may be a bit problematic. On the other hand, Deus Ex Machina lends itself easier for installation than JFLAP for students not familiarized with Java-based applications.

Although Deus Ex Machina provides more different computation models than JFLAP (including linear-bounded automata and register machines), JFLAP provides facilities for integrating automata theory with their corresponding grammars. This demonstrated to be very motivating and helpful for our students.

[2] See `http://www.cs.duke.edu/~rodger` for details

Finally, JFLAP makes use of different colors to highlight nodes, arcs, input tape, among other features. In contrast, DEUS EX MACHINA has a much more 'pale' outlook, as colors are basically used for highlighting nodes as step-by-step execution is performed.

3 Single-purpose software tools for *FL&AT*

In addition to the major software packages discussed in the previous section, there exist a number of different simulation programs intended as a teaching aid for specific topics in *FL&AT*. Next we will briefly describe some of them as well as their main features.

3.1 Formal Languages

Besides JFLAP (which provides a natural connection between automata theory and their associated grammars), other specific tools for dealing with specific topics of formal languages have been developed.

PÂTÉ is a visual and interactive tool for parsing and transforming grammars. PÂTÉ can show the textual or graphical visualization of a derivation for a given grammar (restricted or unrestricted). With the textual visualization, a step-by-step derivation is displayed including the rules used at each step. In the graphical visualization, a parse tree for the derivation is shown with each node representing a symbol or a variable. PÂTÉ can also deal with grammar transformations of context-free grammars, by successively performing removal of λ productions, removal of unit productions, removal of useless productions and conversion to CNF. In these transformations, a graphical representation is used to help the user determine new productions in the removal of unit and useless productions.

PÂTÉ is complemented by PUMPLEMMA, a visual tool for applying the pumping lemma for regular languages. The typical analysis of cases which leads to determine that a language L is not regular via the pumping lemma can be performed using this software.

It must be remarked that both PÂTÉ and PUMPLEMMA were also developed by Susan Rodger and her group at Duke University.

3.2 Finite State Automata

Many simulation programs for FSA have been developed, ranging from Java applets executable from a Web browser to standalone application programs. We will briefly describe some of those we found particularly interesting for *FL&AT* courses:

- CAVE (Constructive Algorithm Visualization Environment) was designed to illustrate the operation of several useful constructive algorithms. It is intended to serve as a teaching tool for computer theory instructors and students. More specifically, CAVE implements constructive algorithms for building union and product (concatenation) machines for DFAs (deterministic finite-state automata). CAVE can also produce the complement of a source DFA, and build an intersection machine for two source DFAs. CAVE was developed by Michael S. Tashbook, and it is freely available as a Java-based application from `http://www.cif.rochester.edu/~samurai/cave.html`.

- The Finite State Machine Explorer (FSME) is an interactive graphical system which supports the construction of FSAs. Animation is used to dynamically illustrate their

behavior on given inputs. Furthermore, manipulation facilities are developed to support the study and exploration of automata, including the conversion between equivalent classes of machine, and the automatic generation of layouts. Finally, input and output capabilities are provided to enable the continued development and reuse of constructed machines. The FSME was developed by Matt Chapman from the Dept. of Computer Science at the University of Warwick (UK), and it is freely available at `http://www.belgarath.demon.co.uk/java/fsme.html`.

- ATE is a software tool designed to define and run a Moore transducer automaton (MTA). It is a standalone program which can be run on a PC, allowing the user to define and execute an MTA. The defined MTA transforms a given input string into an output string according to the automaton definition. ATE was developed by Agustín Esmoris from the Department of Computer Science and Engineering at the Universidad Nacional del Sur (Bahía Blanca, Argentina). The executable version of ATE is freely available from `http://cs.uns.edu.ar/~cic/fcc.htm`.

3.3 Pushdown Automata

There are not many specific teaching aids for simulating pushdown automata. Some simulators for PDAs are included in larger software packages (such as DEUS EX MACHINA or JFLAP).

A very interesting teaching tool is IPAA (Interactive Pushdown Automata Animation) [McD01], which provides a Java-based program for designing and running PDAs. IPAA was developed by Jennifer McDonald (Northeastern University, USA), and it includes several useful *views* for a given PDA. Students can specify a PDA by using an icon-based interface, drawing the PDA as a graph. The PDA can be then run at different speeds. The input tape and the stack are shown as the computation proceeds.

The executable version of IPAA as well as its source code is freely available from `http://www.jenimac.com/JPT/Automata/details.htm`.

3.4 Turing Machines

There are many computer simulators for Turing machines available on the web, most of them available as Java applications that can be run from a browser. In our experience, the Applet Turing Machine Simulator written by Suzanne Skinner[3] has proven specially successful. This applet has several sample Turing machines which can be run at different speeds for arbitrary input strings. The applet allows the student to modify the behavior of a given Turing Machine by providing new tuples or changing existing ones. Turing machine programs (set of tuples) can be loaded from / saved onto disk. Turing machines can also be run in a step-by-step fashion.

Some of the attractive samples included in this simulator are a palindrome detector and 'busy beaver' functions (for 5 and 6 states). Busy beavers are Turing machines with a fixed number of states which are thought to be write as many symbols as possible on the input tape and stop after a given number of steps. These 'busy beavers' are quite puzzling for students as they are rather simple Turing machines which run for a considerable time

[3]This applet is available from many websites around the world. We made it available for our students at `http://cs.uns.edu.ar/~cic/fcc.htm`.

before stopping. (e.g. a 5-state busy beaver can write 4098 symbols before coming to a halt). Finding new busy beaver functions which outperform existing ones has proven to be an attractive challenge for some students.

A more sophisticated tool is provided by VISUAL TURING, a graphical IDE that can be used to edit, define and 'play' with Turing machines. It features an advanced graphical editor (including cut copy& paste facilities, multiple undo, etc.). Machines can be run at full speed or step by step, and debugged using breakpoints and watches for variables. The binary executables of this software are freeware and available from `www.cheransoft.com /vturing/download.html`.

3.5 Petri Nets

In contrast with other kinds of automata which are only studied from a theoretical point of view, the design and use of Petri Nets is an active research area with several industrial applications. Many simulators for Petri nets can be found on the Web, most of them intended to illustrate only the basic aspects of Petri nets. An interesting basic Petri Net simulator was developed by Patrice Torguet. It provides the basic facilities for specifying and executing a Petri Net in a step-by-step fashion. This simulator is available at `http://cs.uns.edu.ar/~cic/fcc.htm`.

For teaching purposes we found the SIMPRES simulator developed by L.A. Cortés as the best alternative for our students. It is a model intended to represent embedded systems, which extends Petri nets adding data and real-time information to tokens, and associating functions and delays to transitions. SIMPRES is a simulator for a subset of this particular computational model. The class of systems that may be validated using SIMPRES corresponds to time Petri nets. The SIMPRES simulator is freely available at `http://www.ida.liu.se/~luico/`.

4 Some Design Rationales for software tools in FL&AT

From our experiences using simulators as teaching aids to complement the different topics presented in a *FL&AT* course, we have identified a number of features which, in our opinion, are of interest for developing this kind of tools:

- **Provide facilties for full interactivity:** As emphasized in [McD01], students should be able to design, modify and test their own automata at their own pace. They should also be able to modify sample automata provided by the teacher. Such change-and-test process proved to be very useful and motivating (e.g. in the case of sample Turing machines or sample Petri nets). Starting from an initial set of sample automata helps students to get comfortable with the tool before they begin a self-guided exploration with other problems.

- **Visualize algorithms whenever possible:** According to our experiences, animation and visualization of algorithms is another important issue when assessing the impact of a simulator as a teaching tool. Earlier versions of computer simulators (e.g. for finite state automata or Turing Machines) were only text-based, forcing the user to define his/her automata using a script-like language. Most current versions we have tested make a strong use of visual elements which lend themselves natural

for simulating algorithmic processes. These computer simulators can be often run as Java applications which are platform-independent requiring only a web browser.

- **Illustrate theoretical properties with built-in features:**

  It must be stressed that several concepts introduced in a *FL&AT* course are constructive proof procedures, so that the same steps can be recast and shown on the screen in a step-by-step fashion (e.g. Kleene's theorem, FSA minimization, etc.) In this respect the JFLAP framework proved to be an excellent example of how to relate different kinds of automata in terms of such theoretical properties.

- **Allow different views for the same problem:** As already discussed in section 3.3, having different *views* for simulating a given automaton has a strong pedagogical impact. In the IPAA simulator for pushdown automata [McD01] four different views are proposed: *tape view* (showing the current input tape), *stack view* (showing the current stack), *path view* (verbose mode) and *automata image view* (PDA visualization as a graph).

  In our opinion, these four views can be generalized to consider different classes of automata presented in a *FL&AT* course. Next we discuss some guidelines for considering these elements in a more general setting:

  - **Input/Output View:** In the case of FSA, PDA or Turing Machines, the basic input view is provided by visualizing the *tape*. onto which the input string is written. In the case of Turing machines, the same tape is used as an input/output device.

    Tapes are easy structures to model as they behave like sequential files. Most students taking a *FL&AT* course have already worked with sequential files in a Pascal-like language, so they should be able to think about a tape as such a file. Hence it is important to borrow the same visual conventions for representing sequential files in order to represent tapes (e.g. the representation of the current cell/record to be processed, etc.).

    Other automata (such as Petri nets) are not tape-based. In such cases a clear representation of input and output values is also needed. In the case of Petri nets, most computer simulators represent tokens in places as circles or numerical values (corresponding to the numbers of tokens in a given place). Petri nets can also be used as language recognizers by labeling transitions with symbols in a given alphabet. In that case, transitions being fired provides an output for the Petri net similar to a production rule. No computer simulator was found to depict transitions in this way.

  - **Storage (or auxiliary data) View:** Some automata (such as pushdown automata) rely on particular storage structures when solving problems (e.g. a stack). In other cases (such as Turing machines) it may be useful to have 'snapshots' of every instance of the tape after each single step executed by the automaton when it was run. Such snapshots can help understand the way computation proceeds in complex settings (e.g. Turing machines with multiple tapes). Auxiliary structures as the ones described above should be visible and properly updated as the automaton is being simulated.

– **Path view:** In this view the student should be able to read a natural language description of every action performed by the automaton with a given input string. Two possibilities should be taken into account:

1. *Basic path view*: every action is shown as an item in a list. The item consists of a term explaining the basic meaning of the action. E.g: State s_0, read 'a'. State s_1, read 'b', etc.
2. *Full path view*: actions are described in a full-fledged text. The student would get a detailed explanation of every step performed during the execution (as if he were being told about this explanation by a human assistant). E.g:
   First the input string is aabbcc.
   Current symbol is "a"
   Current state is s_0.
   After processing symbol "a", next state is s_1,
   and current input string is "abbcc".
   Now current state is s_1.
   ...

– **Image view:** This view should provide a graph-like representation of the automaton the same way the student would see it on a blackboard. After building the automaton the student should be able to run it by providing an arbitrary input string. As the automaton is being run, the current states and arcs should be easily recognizable (e.g. by highlighting them). States and paths already visited should be distinguished from those that were not yet reached (e.g. as done in [McD01]). Such conventions are intuitively helpful for identifying loops and unreachable states.

– **Zoom in/out view:** Features for zooming in and out all relevant information for a given automaton should be provided. Some simulators (e.g. DEUS EX MACHINA or VISUAL TURING) provide ways of invoking 'sub-machines' (sub-automata) for solving a problem. This favors a modular approach (playing the role of subprograms in a Pascal-like programming language), making easier for the student to cope with complex problems.

5 Conclusions

Most topics in the *FL&AT* curricula rely on very simple but abstract theoretical concepts. Theoretical computer simulators as the ones analyzed in this paper can provide an interesting link between theory and practice. In our opinion, *FL&AT* simulation environments should reinforce the significance of theoretical issues when solving practical exercises (as done in DEUS EX MACHINA when requiring a full formal definition of the input language *before* starting to define an automaton). Otherwise many students tend to skip the formal side of the problem, focusing only on its algorithmic aspects.

According to our teaching experience, most topics in the *FL&AT* curricula can be illustrated using such simulators, resulting in a very motivating approach for the students. Nevertheless, students should always be warned that the focus of the course is *not* on mastering the simulators themselves, but rather on using them as a visual tool to aid during the learning process.

It must be remarked that getting familiar with any simulation environment is time consuming for all students. In this respect multi-purpose simulator programs (such as DEUS EX MACHINA and JFLAP) are a good choice as they provide a unified view of different kinds of automata. However, we found that many students were quite enthusiastic to try different simulation programs rather than concentrating in a single one. In our experience, the existence of a bunch of different computer simulators for the *same* automaton proved to be an additional motivation.

Acknowledgments

The authors want to thank William Yurcik (Illinois Wesleyan University, USA) for providing valuable information about different computer architecture simulators.[4] His excellent work on web-based resources for this topic was the inspiration for this paper. We also want to thank Jürgen Dix (University of Manchester, UK) for his comments on the use of busy beaver functions as additional motivation when teaching Turing machines.

References

[Aug95] Juan Carlos Augusto. *Fundamentos de Ciencias de la Computación - Notas de Curso*. Universidad Nacional del Sur, Bahía Blanca, Argentina, 1995.

[GR99] Eric Gramond and Susan Rodger. Using jflap to interact with theorems in automata theory. *SIGCSE Bulletin*, pages 336–340, 1999.

[LP98] Harry Lewis and Christos Papadimitriou. *Elements of the Theory of Computation (2nd. Edition)*. Prentice Hall, 1998.

[McD01] Jennifer McDonald. Interactive pushdown automata animation. *SIGCSE Bulletin*, (1):376–380, 2001.

[Tay98] Gregory Taylor. *Models of Computation and Formal Languages*. Oxford University Press, 1998.

[YG02] William Yurcik and Edward Gehringer. A Survey of Web Resources for Teaching Computer Architecture. In *Proc. of the Workshop on Computer Architecture Education (WCAE 2002), Anchorage AK, USA*, 2002.

[Yur01] William Yurcik. The Simulation Education Homepage. *Simulation: Journal of the Society for Modeling and Simulation Intl.*, pages 202–206, April 2001.

[YYP01] Cecile Yehezkel, William Yurcik, and Murray Pearson. Teaching Computer Architecture with a Computer-Aided learning Environment: State-of-the-Art Simulators. In *Proc. of the International Conference on Simulation and Multimedia in Engineering Education (ICSEE 2001) Phoenix, Arizona*, 2001.

[4] See `http://www.sosresearch.org/caale/caalesimulators.html` for details.