MTSolution: A Visual and Interactive Tool for a Formal Languages and Automata Course

Mariano Martínez
mmartine@exa.unicen.edu.ar

Rosana Barbuzza
rbarbu@exa.unicen.edu.ar
PLADEMA - ISISTAN

María Virginia Mauco
vmauco@exa.unicen.edu.ar
INTIA

Liliana Favre
lfavre@exa.unicen.edu.ar
INTIA

Departamento de Computación y Sistemas
Facultad Cs. Exactas
Universidad Nacional del Centro de la Pcia. de Buenos Aires
(7000) Tandil, Buenos Aires, Argentina

Abstract

There is a need to introduce Information Systems students to advances in languages and automata theory in the early stages of their formation. Visualization and interactivity allow students to play an active role in the learning process, experimenting with the concepts to receive feedback. For this purpose, we propose MTSolution, an educational, visual and interactive software tool that allows teachers and students to experiment with different kinds of abstract models (automata, grammars and regular expressions). With this tool, students can improve their understanding and self-evaluate their own skills designing and testing models. In particular, MTSolution supports the concept of sub-machine providing a library of Turing machines that can be reused in modular designs. MTSolution is based on a client-server architecture and it is implemented in Microsoft Visual C^{++}.NET.

Keywords: software tools, Automata, Turing Machine, grammars, formal languages

1. INTRODUCTION

The emergence of new tools, techniques and paradigms forces a continuous re-evaluation of the topics covered and teaching-learning didactic strategies used in IS educational curricula (Armoni, 2006; Chesñevar, 2004). Modern IS students should understand the basis for IS modeling as well as the pragmatical implications of impossibility and intractability results.

With the growth in volume of online data, for example in databases and on the Internet, the focus of research in information retrieval has shifted to new applications in information management and decision support that demands asymptotically efficient algorithms. Activities such as Data Mining, Data Warehousing, Latent Semantic Indexing, and On Line Processing open directions for research in combinatorial algorithms, models, complexity and computability. In

addition, modeling languages such as the Business Process Modeling Notation (BPMN), the Unified Modeling Language (UML), and the Business Process Execution Language (BPEL) are based on concepts from automata theory. Modeling behavior among components in UML requires designing interaction diagrams, state diagrams, or activity diagrams which are all based on automata theory. Other applications are related to modeling of concurrency and synchronization of processes, and cellular automata, among others.

The Undergraduate Degree Program in Systems Engineering in our career has incorporated, since 1996, "Computer Science I" as an introductory course in the theory of formal languages and automata (FLA) (Favre, 2000). The main purpose of this course is to present an introduction to the study of computational processes and to explore their scope in the context of an automata and grammar hierarchy, with a suitable approach for first-year undergraduate students. The intention is to provide insights on essential questions about the nature of computation: What is an algorithm? What can be computed? When is a given algorithm intractable?

At first, as happened with traditional FLA courses, we used lectures and pencil-paper problem solving approach to teach course contents. From our teaching experiences, we noticed that students were not as motivated and interested as in programming courses, though they could understand the concepts. Then, we considered the inclusion of a software tool to experiment with FLA concepts, as interactivity and visualization are keys to motivate and improve understanding. Several tools have been developed for experimenting with automata and grammars, and most of them are freely available via the Internet. However, almost all have been designed as tools for more advanced students (Rodger, 2004) or to cover a subset of FLA concepts (Barwise, 2005; Grinder, 2003; Forlan Project, 2007; White, 2006).

We then developed MTSolution, an educational, visual and interactive tool that can be used as an aid in learning the basic concepts of FLA theory and consolidating knowledge inside and outside classroom. This tool integrates automata and grammars with an appropriate approach for first-year undergraduate students, who have a wide diversity of theoretical background and prior computing experience.

In this paper, we describe the architecture and functionality of MTSolution. MTSolution design differs from previously mentioned tools in that it is based on a client-server architecture, thus allowing students to practice either on a single computer for homework assignments, or in a network environment for computer labs. MTSolution allows one to define and manipulate different kinds of abstract models such as automata, regular expressions and grammars. It may be used to design and simulate deterministic and non-deterministic versions of finite automata (FA), pushdown automata (PDA) and Turing machines (TM). MTSolution also supports the concept of sub-machine providing a library of TM that can be reused in modular designs, thus linking and emphasizing crucial concepts of software reuse and modularity.

MTSolution has been used along the course and it has proven to be a motivating link between theory and practice, covering most of FLA concepts. The tool is easy to download and install, and it is available in (Martínez, 2007), in Spanish and English. This web site also contains a full description of the tool as well as step-by-step development of some examples.

This paper is structured as follows. Section 2 describes the architecture and components of MTSolution, exemplifying FA, PDA, TM, and grammars. In Section 3, we present the evaluation of the use of MTSolution in a FLA course. Finally, conclusions and future work are mentioned in Section 4.

2. MTSOLUTION

MTSolution is an educational, visual and interactive tool based on a client-server architecture that can be used as an aid in learning the basic concepts of FLA theory. Working in a didactic, visual, interactive, more intuitive and friendly environment users can design, debug and run different types of deterministic and non deterministic finite state machines (FSM), recognizers or transducers, and experiment with grammars and regular expressions (RE). With MTSolution users can create FA, PDA, and

multi-tape composite TM and execute them on arbitrary input strings, in continuous or step-by-step mode, receiving immediate visual feedback. They can also work with RE, regular (RG), context-free (CFG) and context-sensitive (CSG) grammars randomly showing strings they derive or deciding if a particular string may be derived by the grammar or not. In addition, MTSolution assists users in the conversion of FA, RE and RG between each other.

Architecture

MTSolution is based on a client-server software architecture. Clients and servers may be placed independently on computers in a network, possibly on different hardware and operating systems. Generally, clients rely on servers for resources or processing power (in our case, fast server and workstations or PC's for students). However, in MTSolution a single computer can be both a client and a server depending on the software configuration. Thus, MTsolution can be used either to practice with a single computer for homework assignments or to assist in classrooms in a network environment.

MTSolution server executes FSM and simulates parsing and generation of strings as requested by clients, while clients interpret the results returned by the server. Basically, this model allows the separation of functionalities for string recognition and user interface, encouraging incremental and easier development and testing, and better maintainability of client and server modules. By providing a protocol of communication between client-server modules, it is possible to design FSM editors in different platforms, and languages, among others.

Figure 1 describes the architecture of MTSolution. *MTServer* is the application server; its main functions are the simulation of FSM execution and string parsing and generation. Before execution, *MTServer* transforms each FSM into a TM. *MTEditor* is the FSM editor and it allows creating, saving, and recovering recognizer or transducer FA, PDA and TM. It contains several special algorithms to work with regular languages (RL). In addition, to design transition diagrams for FA and PDA, we used the framework Unidraw (Vlissides, 2007) which offers many facilities to create graphical editors.

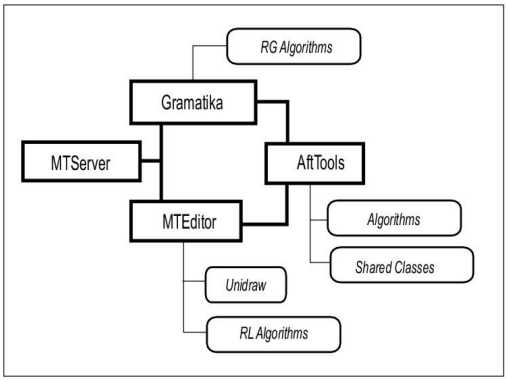


Figure 1 - MTSolution architecture

Gramatika is the editor of grammars. It provides functions for editing, saving and recovering definitions of RG, CFG, and CSG (according to Chomsky hierarchy). It also has some special algorithms to transform RG into FA. Besides, it allows the random derivation of strings in the language of the grammar or the verification of membership of a string to the grammar language.

AFTools is a Dynamic Linking Library (DLL) which contains common and reusable components and functions for the different MTSolution applications.

Main features of MTSolution

The following are some of the features that make MTSolution a helpful tool for teachers and students, especially for beginner CS students:

- *Formal definition of abstract models:* MTSolution encourages students to understand the importance of formalism. For example, when designing automata or grammars the first step is the definition of the alphabets to be used. In addition, the tool automatically completes, updates and shows the formal definition of FA, PDA, and TM throughout their creation or modification process. It also shows error messages when it detects an invalid situation, e.g. when trying to define two initial states for a FA or using the same symbol as terminal and nonterminal in the definition of a grammar.

- *Step-by step visualization of conversion algorithms:* MTSolution shows the step-by-step execution of the algorithms to

transform FA into RE or RG, RE into FA, NDFA into DFA, DFA into DFA with minimal states, allowing in some cases user's intervention. For example, when converting a FA into a RE the student may choose, in each step, the state to remove, thus helping students to check the result previously obtained using pencil and paper with the solution provided by the tool.

- *Definition of Transducers:* The inclusion of these kinds of machines that produce an output associated with an input, helps to show several concrete applications that can be modeled with FA, PDA and MT, thus motivating students to learn and use abstract models. For example, using MTSolution students may solve exercises such as substituting each occurrence of two or more blanks in a text file with a single blank, adding two binary-coded natural numbers, calculating the number of occurrences of a given string in a text file or modeling a vending machine which dispenses candy bars or drinks, among others.

- *Composite parameterized multi-tape TM:* The natural way to construct a complex TM is to start from simpler and more reusable ones, as happens when developing a complex program. MTSolution provides facilities to compose TM. This way of designing TM allows us to emphasize the important and recurring concepts of abstraction, reusability and modular design introduced in programming courses.

- *Step-by step execution of machines:* Recognizers and transducers FA, PDA and TM may be run at full speed or in a step-by-step mode. The last option allows users to step through transitions one at a time, highlighting the current transition and showing the input string already scanned. In this way of execution, the instantaneous description of the machine is shown, giving users the possibility of following exhaustively the recognition process of a string. This helps users to better understand the machine behavior, thus detecting and correcting errors.

- *Facilities for simulation and definition:* Input strings to test machines or grammars may be provided interactively or may be loaded from a text file. Both possibilities are also available for definition of alphabets (input, stack, tape). This allows one not only to reuse alphabet definitions but also to provide students a set of input strings to test correctness of machines and grammars.

- *Informative error messages:* When MTSolution detects a mistake, it shows an appropriate error message connecting the theoretical concepts with the concrete problem in order to guide students in correcting the error. This is mainly important when students are working on their own and do not have a real teacher to assist them.

- *Random generation of strings:* In contrast with traditional parsing, which given an input string and a grammar tries to decide if the string is in the language of the grammar or not, this functionality shows grammars as generative devices. Taking as input a grammar and a length, MTSolution randomly generates strings, no longer than length, that belong to the language of the grammar. For each generated string, it shows the corresponding derivation and the productions used.

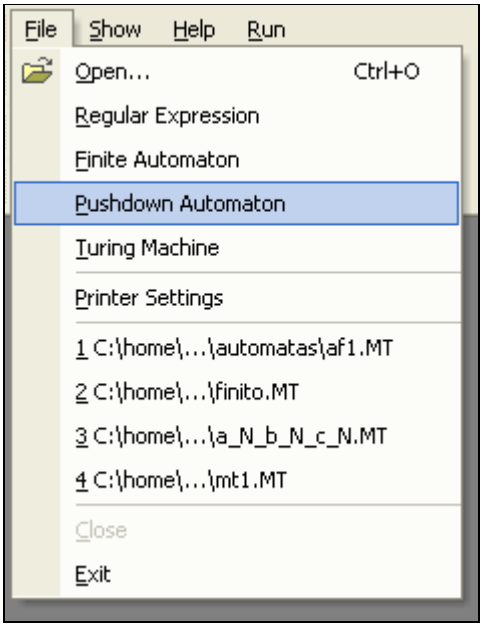


Figure 2 – File option in MTEditor menu

MTEditor

MTEditor, the automata editor, is basically composed by a main panel, a menu bar and toolbars that allow users to easily edit, save, recover and run FA, PDA and TM (Figure 2). Recognizer or transducer FA or PDA can be created on a simple canvas, by drawing

states and transitions at chosen locations. Multi-tape and composite TM can be defined by giving the transition table. FA, PDA and TM may be run with arbitrary input strings in order to answer if the string is accepted or not by them, in case of a recognizer one, or to return the corresponding output string, if executing a transducer. The automata design, editing and executing interface is modern, simple, intuitive, friendly, and easy to understand and use.

MTEditor also contains algorithms to perform the following transformations: non deterministic finite automata with ε-transitions (NDFA-ε) to non deterministic finite automata (NDFA), NDFA to deterministic finite automata (DFA), DFA to minimal state DFA, FA to RE or RG and RE to NDFA-ε.

Next, we present three examples to illustrate FA/PDA design and execution, and composite TM definition and simulation. Later, we give some examples of the application of the algorithms on FA provided by MTEditor.

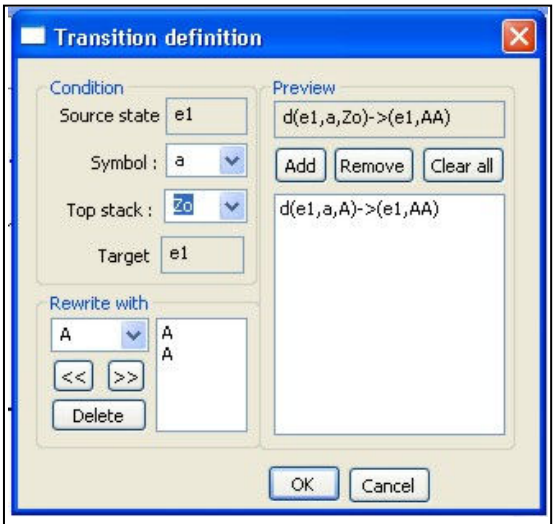


Figure 3 - Definition of transitions in PDA

Finite and Pushdown Automata: To define a new FA/PDA the corresponding option from the main menu should be selected. First, input alphabet should be defined; in case of PDA definition, stack alphabet should be also specified. Next, states and transitions may be added by using a mouse to draw states on a scrollable area, clicking and connecting states with transitions, and right clicking on any component to obtain a context menu to modify its properties, such as state name, initial or final state, or transition label. Figure 3 shows how to specify a transition in a PDA. The transition table is dynamically built by MTSolution and it is available by clicking the tabbed pane Table (Figures 10 and 11, Appendix).

Once completed the FA/PDA definition, it may be executed on an arbitrary input string. Figure 10 shows a stepwise execution of the FA recognizing the language L = { x / x ∈ {a, b}* and (x contains substring aa or x contains substring ba)}, given the string *abaa*. Figure 11 shows a stepwise execution of the PDA recognizing, by final state, the language L = {$a^n b^n$ / n > 0}, given the string *aaabbb*. For both automata the input string may be provided interactively or loaded from a text file, and it should be entered in the tabbed pane Tapes which, in these cases, allows the definition of only one tape, the input tape. MTEditor sends to MTServer a request of execution with the FA/PDA already defined and the given input string.

For an accepted string, MTServer returns the sequence of transitions followed by the FA/PDA to recognize the string. Then, MTEditor uses this transition list to display, for example, a step-by-step execution; the current transition is highlighted (in Edition and Execution panels) as the user goes forward, and the symbols of the string already scanned are shown in the Execution panel (Figures 10 and 11). In case of PDA execution, the stack content is also shown.

Composite Turing Machine: One of the main features of MTSolution is the possibility to design composite TM by combining simpler ones. Each basic machine may be defined as a parameterized multi-tape TM, with parameters passed by reference. We detail below the construction of a composite TM to add two unary-coded natural numbers (Figure 12, Appendix). The first number is located in tape M, the second one in tape N, and the output number is left in tape M+N. We use three basic sub-machines. The first one (Init.MT) initializes the tape, writing the symbol X in a cell and leaving the read/write head in the next cell. The second TM, Copy.MT, copies the whole content of its first parameter tape to the second parameter tape. Finally, Back.MT rewinds the parameter tape until it finds the symbol X, and then it moves right.

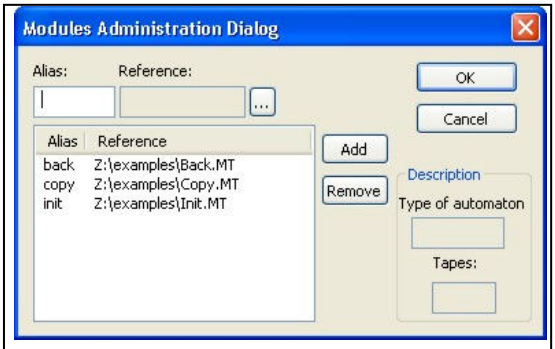


Figure 4 - Definition of alias for TM

Sub-machines are invoked by the composite TM by using an alias name. Figure 4 shows how to associate each .MT file with the corresponding alias. In this example, we use the name of each file as alias.

These sub-machines may be used instead of a target state in the definition of the composite TM (Figure 12). Each call is defined as the symbol "@" followed by the alias name, and the parameters. The first one is a return state which will be used to perform a return to the composite TM in case the sub-machine ends in its final state; the second one is also a return state but to be used in case the sub-machine ends in a non final state. The third parameter is a list of tape names of the composite TM to be used by the sub-machine. MTSolution includes an assistant to guide users in the definition of sub-machine calls.

The composite TM (add.MT) in Figure 12 proceeds as follows: it calls the sub-machine Init.MT to initialize M+N output tape. On return, if one of the numbers is greater than 0, the composite TM goes to state e_1, to begin the addition of numbers; in other case, it goes to state e_3 and finishes. In state e_1, if number in M is greater than 0, it calls Copy.MT sub-machine to add M number to M+N tape; the same happens with N number if it is greater than 0. In both cases, the return state is the final state of the composite TM (e_2). When both numbers are greater than 0, the composite TM first adds M number to M+N output tape, by calling Copy.MT, and on return it calls Copy.MT again in order to add N number to the M+N output tape. On return, it goes to the final state.

Figure 12 contains the definition and one possible execution of the composite TM with input numbers 3 and 2 unary-coded, displaying the contents of each tape and the current position of the read/write head of each tape by highlighting the corresponding cell.

Algorithms: As we had already mentioned, MTEditor contains some algorithms that can be executed on FA. The option Algorithms, in the main menu, gives the possibility of choosing the desired transformation to be applied to a FA (Figure 5). It is important to remark that for all transformations not only the final result is shown but also how it was obtained, as we explain below. The aim is to help students to follow and test their own results and the steps performed to get them.

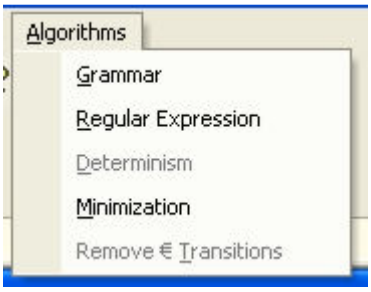


Figure 5 – Algorithms option in FA MTEditor menu

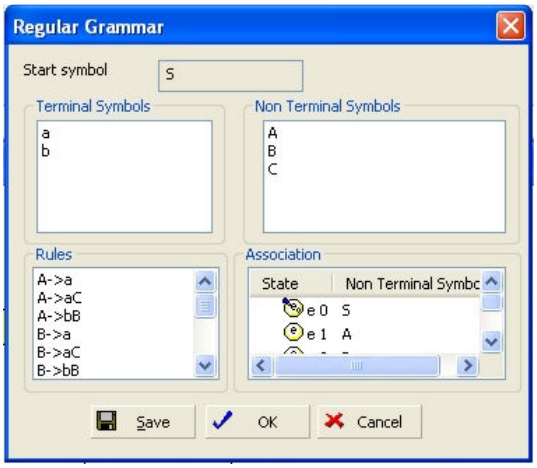


Figure 6 – RG derivation from FA

For a concrete FA only applicable options are enabled. For example, for the FA in Figure 10 the option Deterministic will be disabled because the FA is already deterministic

(Figure 5). For this FA we can derive automatically a RG by choosing Grammar option in the Menu. Figure 6 describes all the components of the grammar derived, indicating how non terminal symbols where associated to FA states in order to apply the derivation algorithm for RG from FA defined in (Hopcroft, 2000). This grammar can be saved in a file to be opened and edited later with Gramatika. For the same FA we can also choose to minimize it or to obtain a RE describing the same language it recognizes.

For a concrete FA only applicable options are enabled. For example, for the FA in Figure 10 the option Deterministic will be disabled because the FA is already deterministic (Figure 5). For this FA we can derive automatically a RG by choosing Grammar option in the Menu. Figure 6 describes all the components of the grammar derived, indicating how non terminal symbols where associated to FA states in order to apply the derivation algorithm for RG from FA defined in (Hopcroft, 2000). This grammar can be saved in a file to be opened and edited later with Gramatika. For the same FA we can also choose to minimize it or to obtain a RE describing the same language it recognizes.

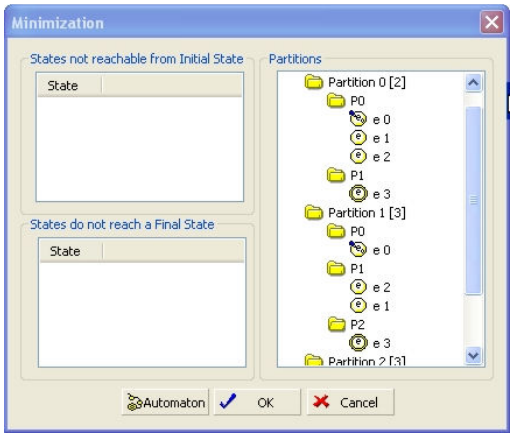


Figure 7 –Information of minimization algorithm for FA in Figure 10

In case of minimization, the result for FA in Figure 10 is shown in Figure 13 (Appendix) while Figure 7 describes the partitions obtained and useless states eliminated based on the FA minimization algorithm proposed in (Aho, 1986) (A state is useless if it does not reach a final state or if it is not reachable from the initial state).

RE derivation is performed by using Thompson´s algorithm (Aho, 1995) and it involves user participation. Users have to choose the order in which they want to eliminate states until they get a FA with two states: the initial and a final one (it may be the case that only one state remains, this happens when the initial state is also a final one). The application of the RE derivation algorithm applied to FA from Figure 10 is shown in Figure 14 (Appendix).

Gramatika

Gramatika is the editor of grammars and it provides functions for editing, saving and recovering definitions of left and right-linear RG, CFG, and CSG. During grammar edition, the application checks if every typed production matches the appropriate production pattern for the grammar in construction, showing error messages in case of discrepancy. Other functions give the possibility of verifying if a given string can be generated by the grammar or not, displaying when possible the corresponding derivation. Besides, Gramatika may generate, in a random way, strings belonging to the language specified by the grammar which are no longer than a length given by the user.

For RG, Gramatika automatically generates the corresponding NDFA, which may be recovered from MTEditor to be transformed into DFA or RE.

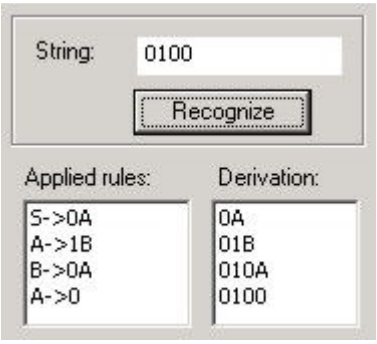


Figure 8 - Derivation of 0100

We show an example of the definition of a right-linear RG for the language:

L = { x / x ∈ {0, 1}* and (x contains substring 00 or x contains substring 11)}, using Gramatika. The first step is the selection of New Right-linear Grammar from

the main menu. Next, terminal and nonterminal symbols should be defined, also indicating the Start Symbol. The last step is the specification of the production set (Figure 15, Appendix).

With a grammar users can verify if an arbitrary string could be parsed or not (Figure 8) or randomly generate strings to test if the grammar only derives strings of the expected language (Figure 9).

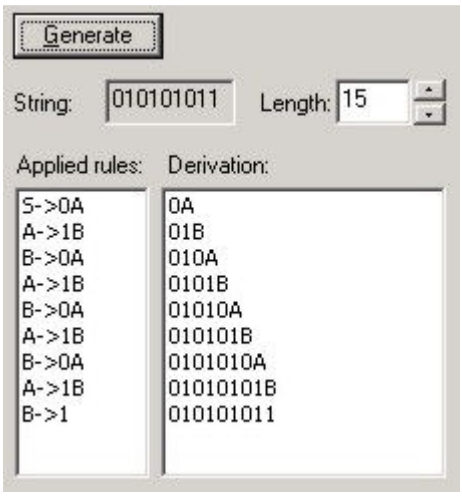


Figure 9 - Random generation of a string

3. EVALUATION

MTSolution has been successfully class-tested since last year. It has been used in lectures, to introduce for example automata design and execution, for homework assignments, and in a computer lab with students and teachers working simultaneously on the same examples. In the last semester, we also used it as a grading tool. We carried out an experience with a small group of 20 students randomly chosen, making them work with MTSolution to solve all the exercises involving FA, PDA, and CFG. Each student of this group had to use MTSolution in a lab to solve the exams given by the teachers to grade students understanding of FA, PDA, and CFG. For example, students were asked to design the automata to recognize languages like $L = \{ x / x \in \{a, b\}^* \text{ and } x \text{ ends in aabaa}\}$, or $L = \{e^{2j} h^{j+1} a^m d^k b^p / p > m; m>0; k, j \geq 0\}$.

As the students could simulate their automata and grammar behavior using the tool, they could test them to correct errors in their designs more easily than the rest of the students, which used paper and pencil. As a consequence, in the first group 90% of the students passed all the exams, while in the other only 60% of the students could succeed. Further evaluation of MTSolution is planned as it will be used in class and as a grading tool for the all the students.

Students report positive experiences with its use (we have approximately 200 students per year). They found MTSolution easy to install and very intuitive to use, and so they were quite enthusiastic in working with the tool on practical exercises. Students also appreciated the assistance provided by MTSolution to correct mistakes when designing automata or grammars.

4. CONCLUSIONS AND FUTURE WORK

MTSolution is a didactic, interactive and easy to use educational tool integrated in an undergraduate course in FLA Theory. This course is a good starting point to make students aware of the basic notions of language processing and computational process limitations in the early stages of their formation. These topics are addressed in a way that emphasizes the reuse of existing TM and the modular design. The principles underlying MTSolution are abstraction, formality, modularity and reuse. MTSolution design is based on a client-server architecture. A small server engine interprets TM and grammars, returning a result to client/s. This software architecture allowed us to follow an incremental implementation and testing. Besides, maintenance, updating, and addition of new components to the tool are easier to perform.

MTSolution was implemented in Microsoft Visual C++.NET, thus improving run-time performance.

MTSolution is useful as a classroom aid tool either to work in a network environment or in an isolated computer.

Teachers and students considered the tool to be easy to teach and learn. In this way, FLA concepts remain in the center with the tool collaborating as an assistant.

We continue evaluating, and extending MTSolution as an integrated educational tool, on the basis of our teaching experience and student suggestions.

We plan to extend the tool to work with closure properties of regular and context-free languages.

REFERENCES

Aho, A., Sethi, R., and Ullman, J. (1986) Compilers: Principles, Techniques and Tools. Addison Wesley.

Aho, A. and Ullman, J. (1995) Foundations of Computer Science, Computer Science Press.

Armoni, M., Rodger, S., Vardi, M., and Verma, R. (2006) "Automata Theory – Its Relevance to Computer Science Students and Course Contents", 37th SIGCSE Technical Symposium on Computer Science Education, pp. 197-198.

Barwise, J. and Etchemendy, J. (2007) Logic Soft from CSLI, http://www-csli.stanford.edu/hp/Logic-software.html (Accessed September 2007).

Chesñevar, C., Gonzalez, M.P., and Maguitman, A. (2004) "Didactic Strategies for Promoting Significant Learning in Formal Languages and Automata Theory", 9th Annual Conference on Innovation and Technology in Computer Science Education, pp. 7-11.

Favre, L, Mauco, M.V., and Barbuzza, R. (2000) "Introducing First-year Students to Theoretical Computer Science", Information Systems Education Conference, ISECON 2000, Philadelphia, EE.UU.

Grinder, M.T. (2003) "A Preliminary Empirical Evaluation of the Effectiveness of a Finite State Automaton Animator", 34th SIGCSE Technical Symposium on Computer Science Education, pp. 157-161.

Hopcroft, J., Motwani, R., and Ullman, J. (2000) Introduction to Automata Theory, Languages, and Computation (2nd Edition). Addison Wesley.

Martínez, Mariano (2007) MTSolution http://users.exa.unicen.edu.ar/$\sim$martine/ (Accessed September 2007).

Rodger, S., Bressler, B., and Finey, T. (2004) Reading, S. "Turning Automata Theory into a Hands-on Course", 37th SIGCSE Technical Symposium on Computer Science Education, pp. 379-383.

The Forlan Project (2007), http://people.cis.ksu.edu/~stough/forlan/ (Accessed September 2007).

Vlissides, J. Generalized Graphical Object Editing (2007)

http://www.ivtools.org/ivtools/unidrawinfo.html. (Accessed September 2007).

White, T., and Way, T. (2006) "jFAST: A Java Finite Automata Simulator", 37th SIGCSE Technical Symposium on Computer Science Education, pp. 384-388.

Appendix

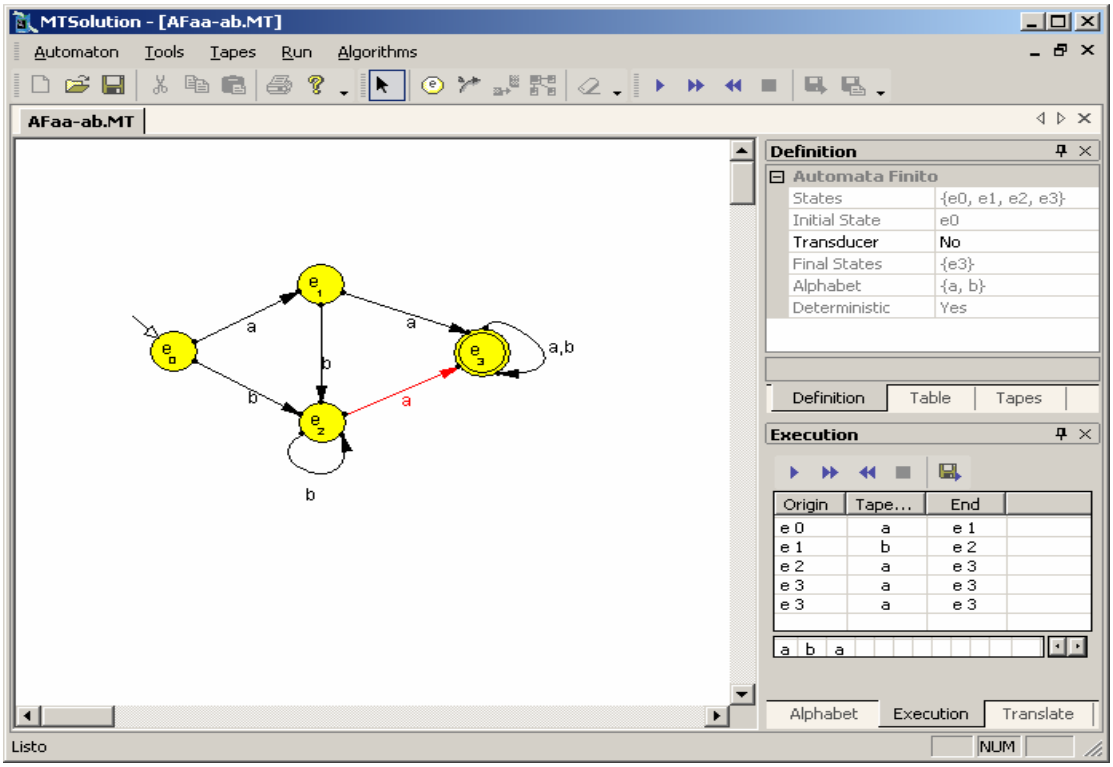


Figure 10 - Example of FA Execution

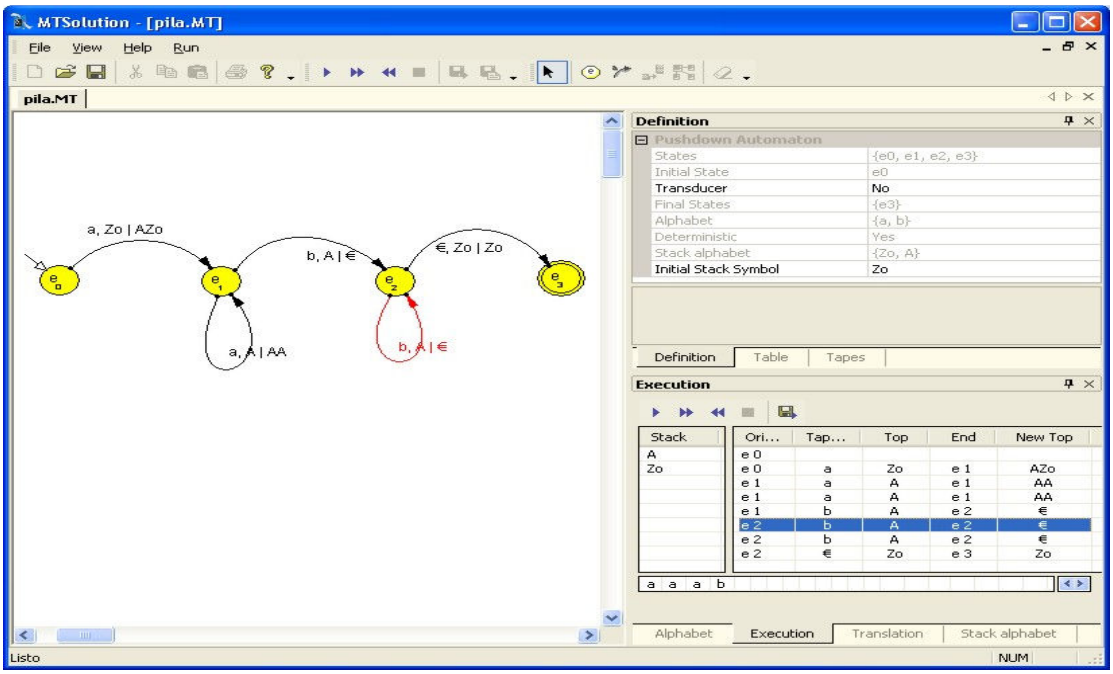
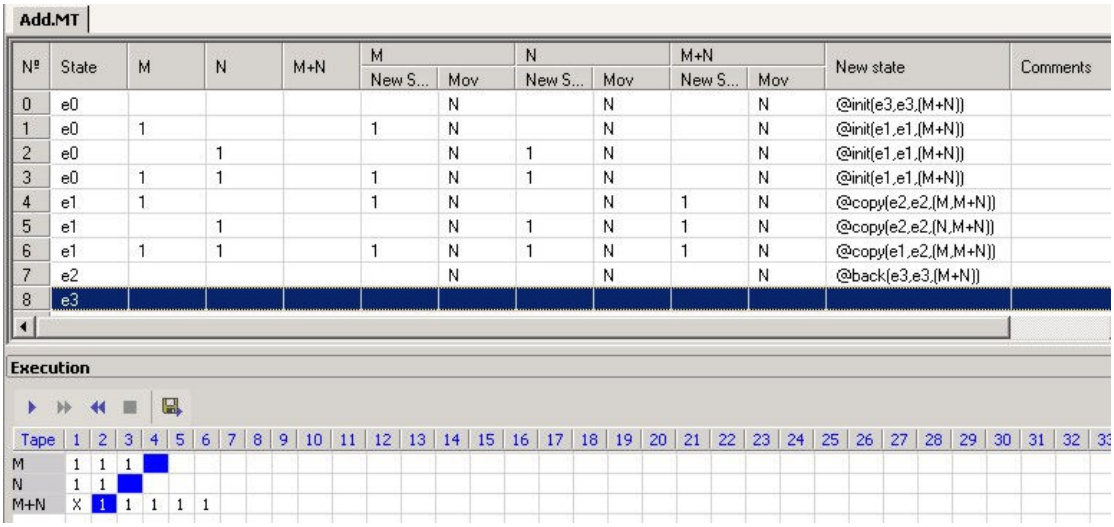


Figure 11 - Example of PDA Execution



Nº	State	M	N	M+N	M		N		M+N		New state	Comments
					New S...	Mov	New S...	Mov	New S...	Mov		
0	e0					N		N		N	@init(e3,e3,(M+N))	
1	e0	1			1	N		N		N	@init(e1,e1,(M+N))	
2	e0		1			N	1	N		N	@init(e1,e1,(M+N))	
3	e0	1	1		1	N	1	N		N	@init(e1,e1,(M+N))	
4	e1	1			1	N		N	1	N	@copy(e2,e2,(M,M+N))	
5	e1		1			N	1	N	1	N	@copy(e2,e2,(N,M+N))	
6	e1	1	1		1	N	1	N	1	N	@copy(e1,e2,(M,M+N))	
7	e2					N		N		N	@back(e3,e3,(M+N))	
8	e3											

Figure 12 - Example of a composite TM

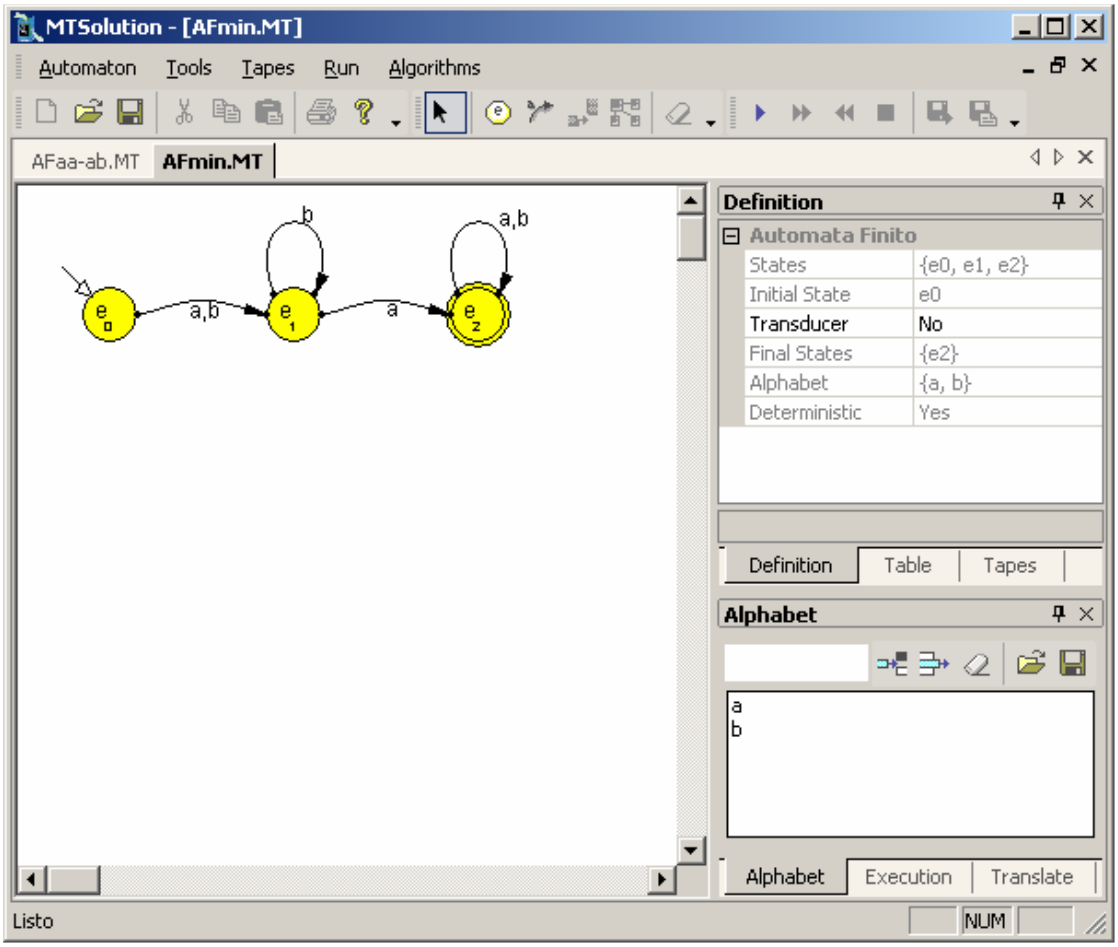


Figure 13 – Minimum FA for FA in Figure 10

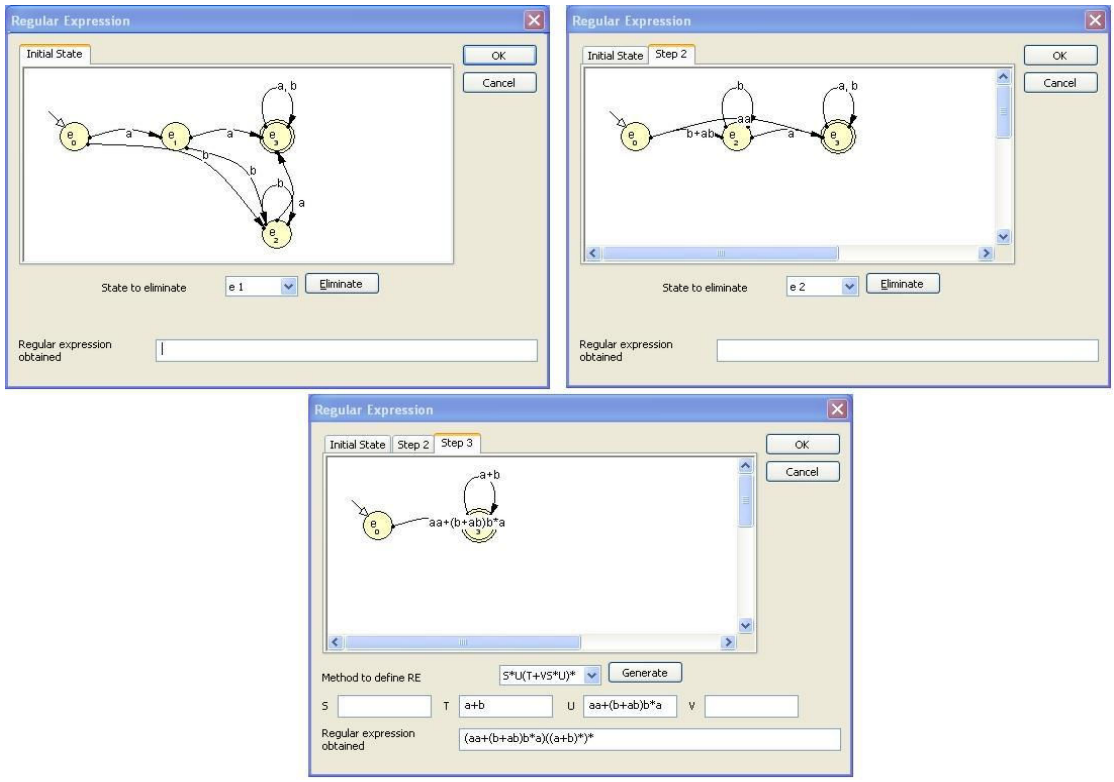


Figure 14 – Steps of RE derivation for FA in Figure 10

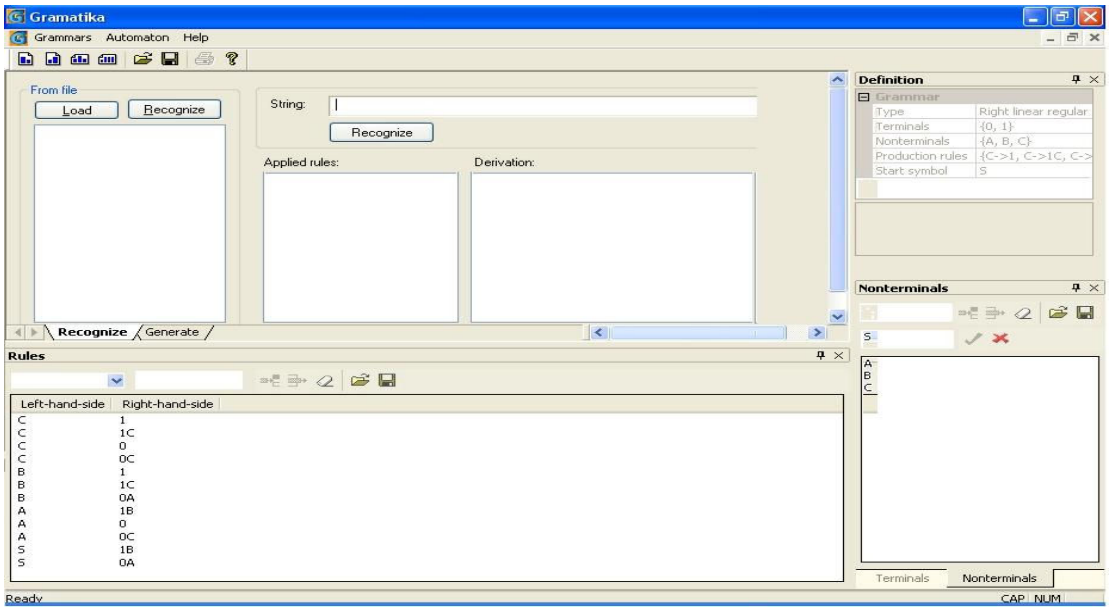


Figure 15 – Definition of right-linear RG