RegExpert: A Tool for Visualization of Regular Expressions

Ivan Budiselic*, Sinisa Srbljic* and Miroslav Popovic*
* University of Zagreb, School of Electrical Engineering and Computing, Zagreb, Croatia,
e-mail: {*ivan.budiselic, sinisa.srbljic, miro.popovic*}*@fer.hr*

***Abstract*—In recent years, tools for computer aided learning have become widespread on all levels of education. These tools are used as replacements or additions to traditional methods of instruction, in order to reduce the time students require to absorb the taught course. On the other hand, these tools ease the burden of course lecturers when preparing complex examples since these can be created and executed in real time and then analyzed in class.**

To increase the interactivity of computer science education, we have developed *RegExpert*, a tool for regular language manipulation and visualization. The main goal of the *RegExpert* tool is to simplify and visually present complex concepts and mathematical models of automata theory fundamentals. *RegExpert* converts user defined or automatically generated regular expression to an equivalent NFA-epsilon (nondeterministic finite automaton with epsilon transitions) and presents it to the user in a form of state diagram. In this paper, we present the structure and implementation of the tool, as well as the ideas for future development.

***Keywords*—computer aided learning, automata theory, regular languages, regular expressions**

I. Introduction

The ever-growing complexity of computer science leaves less and less time in the curriculum to cover fundamentals such as finite state automata and regular languages. The very high level of abstraction of the theory makes it difficult to motivate the students and get them interested in the presented course material. Covering various formal models, such as DFA (deterministic finite automaton), NFA (nondeterministic FA), NFA-epsilon (NFA with epsilon transitions), regular expressions, and regular grammars, is important since each embodies a different approach to the same problem and as such offers a richer toolbox to the student. This only leaves time to cover one or two very basic examples, and these are usually too simple to awe the students or to make them appreciate the importance of the model. The students are then forced to make up example problems of their own, often unable to solve them or even verify their solutions.

This problem is further emphasized by today’s employment policies. Most potential employers are likely to ask the students which programming languages they are familiar with, but not very likely to ask if they understand state machines. However, to fully understand programming languages and how they are compiled into executable code, one must at least have basic knowledge of automata theory, mainly finite state machines and regular languages.

We have developed the *RegExpert* tool to help both students and lecturers. The tool generates regular expressions of different complexity and converts them into corresponding NFA-epsilon. The conversion algorithm used within the tool is explained in every course that deals with automata theory, since it is a proof by construction that every regular expression can be converted to a finite state machine that accepts the language the expression defines [1].

The tool allows students to experiment with various regular expressions and see how the changes made affect the resulting NFA-epsilon. Learning through experimentation is proven to be one of the most efficient ways of education, since the best way to understand an algorithm or method is to see it applied on a live example.

Furthermore, the tool also enables the lecturer to make the lecture more interesting, for example allowing the students to pick the regular expression used for the conversion. The lecturer can also skip the detailed description of every step of the algorithm, since the students can easily analyze the steps from the listing at their convenience. *RegExpert* allows easy construction of different problems of approximately the same difficulty for tests and verification of the students’ solutions.

In Section 2, we briefly elaborate current state of the art in the field of computer aided learning. Section 3 describes the structure of the *RegExpert* tool and discusses the implementation. We conclude the paper in Section 4 giving a summary of the possible applications and future extensions of *RegExpert*.

II. Related Work

Computer aided learning tools are today available for almost any field of science. One of the most popular such tools, used extensively in both education and research, is *Mathematica* [2]. *Mathematica* provides a high-level, interpreted programming language, and offers vast numerical and graphical libraries. The tool also contains *Combinatorica* [3], a collection of over 450 algorithms for discrete mathematics and graph theory.

Simpler, non-commercial tools are also available on the Web. For example, detailed animations of the basic sorting algorithms as well as the source code used can be found at [4]. For computer networks, animations of the various data-link layer protocols can be found at [5].

In the field of regular languages and automata theory, several notable tools exist. *FIRE engine* [6] is a C++ class library implementing finite automata and regular expression algorithms. *Grail+* [7] is a symbolic computation environment for finite-state machines, regular expressions, and other formal language theory objects. Using *Grail+*, one can input machines or expressions, convert them from one form to another, minimize them, complement them, and make them deterministic. *JFLAP* [8] is software for experimenting with formal language topics including nondeterministic finite automata, nondeterministic pushdown automata, multi-tape Turing machines, several types of grammars, parsing, and L-systems. In addition to constructing and testing examples for these, *JFLAP* allows one to experiment with construction proofs from one form to another, such as converting an NFA to a DFA to a minimal state DFA to a regular expression or regular grammar.

RegExpert is not the first e-learning tool developed at the University of Zagreb. Prior to *RegExpert*, we have developed *Automata Simulator* [9] and *SoftLab* [10], both applicable in the field of automata theory fundamentals. *Automata Simulator* offers interactive generation and simulation of finite state machines. It supports all types of finite state machines: automata with binary output (DFA, NFA, and NFA-epsilon) and automata with general output (Moore and Mealy automata). *Softlab* is an extension of *Automata Simulator* towards a fully distributed e-learning tool. It allows better interaction between the student and the supervisor as the supervisor can monitor the student's progress and even assist in modeling more complex examples.

III. The RegExpert Tool

The structure of the *RegExpert* tool is shown in Fig. 1. The tool consists of four main components: the user input/output interface, the regular expression generator, the converter of regular expressions to NFA-epsilon, and the Graphviz. *RegExpert* users input regular expressions either manually or using *RegEx Generator*, which generates regular expressions automatically. The former approach is useful to students for evaluating their paperwork solutions, while the latter approach allows for experimentation with different classes of expressions and helps lecturers to prepare student assignments.

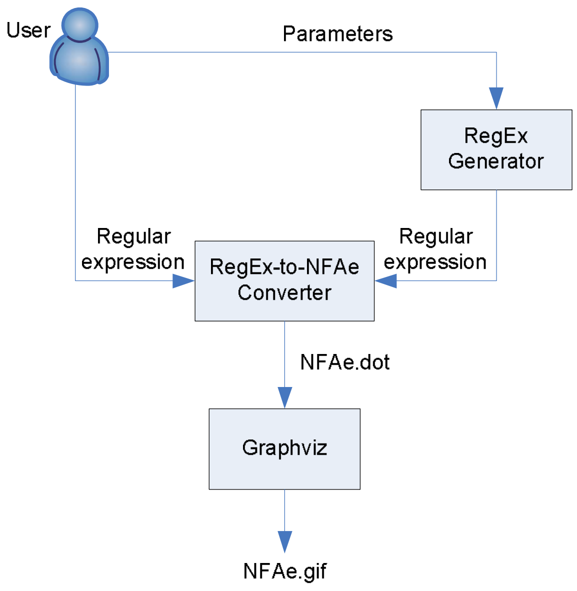
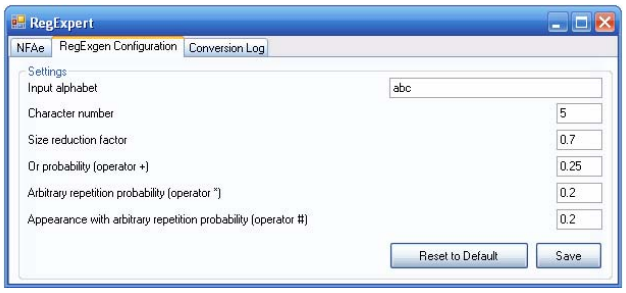


Fig 1. RegExpert structure



Settings

Input alphabet	abc
Character number	5
Size reduction factor	0.7
Or probability (operator +)	0.25
Arbitrary repetition probability (operator *)	0.2
Appearance with arbitrary repetition probability (operator #)	0.2

Fig. 2. Configuration settings for generation of regular expressions

The *RegEx-to-NFAe Converter* transforms the input regular expression to a NFA-epsilon presented in the Graphviz *dot* language [11, 12]. The Graphviz *dot* language is a textual language for representing graphs and was chosen over a classical table representation since it allows for easy conversion to a state diagram. The *dot* language is simple enough to efficiently generate and to read out the table representation if necessary. Finally, Graphviz is used to convert the *dot* file to a gif image that depicts the state diagram of the generated NFA-epsilon.

The format and complexity of automatically generated regular expressions may be controlled using the *RegExgen Configuration* panel, as shown in Fig. 2. Input symbols consist of a single character from the specified *Input alphabet*. The *Character number* option roughly dictates the size of the expression. If the expression is split with the choice operator '+', the left-hand and right-hand expressions get the character number of the whole expression multiplied by the *Size reduction factor*. Therefore, it is imperative to keep the size reduction factor strictly less than one, in order to guarantee algorithm termination. The occurrence of the choice operator '+' is controlled by the *Or probability* setting. The final two options, *Arbitrary repetition probability* and *Appearance with arbitrary repetition probability*, determine the frequency of repetition operators in the generated expression. The standard regular expression syntax for appearance with arbitrary repetition *(expression)*$^+$ is replaced with *(expression)#* to avoid ambiguity with the choice operator '+'.

Fig. 3 presents the main *RegExpert* panel. The '*Generate RegEx'* button invokes the regular expression generator, and writes the generated expression into the text field. Alternatively, the user can edit the text field manually. The *'RegEx -> NFAe'* button converts the regular expression from the text field to the corresponding NFA-epsilon. In the resulting graph representing the NFA-epsilon state machine, epsilon transitions are marked with '\$', the leftmost node of the graph is the starting state of the automaton, while the rightmost node is the single accepting state.

The conversion of the regular expression to the equivalent NFA-epsilon is done using a slightly modified algorithm from [1]. The structure of the state machine is not altered in any way, so the various steps of the algorithm are apparent in the image.

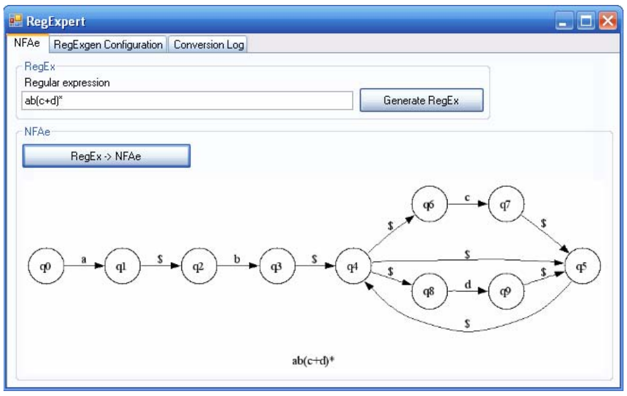


Fig. 3. Main RegExpert panel

The appearance with arbitrary repetition operator ‘#’ is converted to the appearance with an arbitrary repetition operator ‘*’ using the rule

*(expression)# = (expression)(expression)**

If the expression is a single input symbol or the empty string, two states are generated for that expression: a starting state **S**, and an accepting state **A**. A single transition from the starting to the accepting state is added and marked with the input symbol.

If the expression does not contain any top-level choice operator ‘+’, then the sub expressions are processed recursively from left to right. An epsilon transition from the accepting state of the left expression’s NFA-epsilon to the starting state of the right expression’s NFA-epsilon is added. The starting state of the whole expression’s NFA-epsilon is the starting state of the leftmost sub expression’s NFA-epsilon, while the only accepting state is the accepting state of the rightmost sub expression’s NFA-epsilon. In Fig. 3, the expression *ab(c+d)** is separated into sub expressions *a*, *b*, and *(c+d)**.

If an expression does contain a top-level choice operator ‘+’, it is separated into sub expressions that do not. They are processed recursively, and two states **S’** and **A’** are added. For every sub expression, an epsilon transition is added from **S’** to the starting state of the sub expression’s NFA-epsilon and from its accepting state to **A’**. **S’** and **A’** are then made the starting and accepting state of the NFA-epsilon for the whole expression, respectively. In Fig. 3, the expression *(c+d)* is separated into *c* and *d*. States *q4* and *q5* are added and connected to the sub expression’s automata as described above.

Finally, if the expression is followed by an arbitrary repetition operator ‘*’, two epsilon transitions are added: one from the start state to the accepting state and another one in the reverse direction. The epsilon transitions from *q4* to *q5* and vice versa in Fig. 3 are examples of such transitions. The listing of all the important steps done by the program during the conversion can be seen in the *Conversion log* panel.

The graph representing a state diagram of a NFA-epsilon generated by this algorithm is planar, since each of its sub graphs is planar. This enables drawing the state diagram in a screen convenient way, without edge intersections and node collisions. Since the equivalent DFA in general does not retain planarity, this was the primary reason why we have chosen the NFA-epsilon as a visual representation of regular languages.

IV. Conclusion

In this paper, we presented *RegExpert*, a tool for interactive learning of regular languages. The objective of the tool is to make the learning process entertaining and simple for students, while lessening the burden of problem and example preparation for the lecturers. Our experiences from the development of *Automata Simulator* [9] and *SoftLab* [10], and their application in computer science curriculum at the University of Zagreb encourage us to further advance the education by applying the *RegExpert* tool in *Introduction to Theory of Computation* course.

A further step in improving the tool and making it classroom-ready is implementation of conversions between basic models of finite state machines (NFA-epsilon, NFA, and DFA), as well as implementation of DFA minimization. Since drawing nonplanar graphs in a visually pleasant way is difficult to define precisely and for almost any definition the problem is NP-complete, we propose a table form as an initial representation of these state machines.

References

[1] S. Srbljic, *Compiler Design 1: Introduction to Theory of Formal Languages, Automata, and Grammars*, (original title in Croatian: “Jezicni procesori 1: uvod u teoriju formalnih jezika, automata i gramatika”), 2nd Edition, Element, Zagreb, 2002, pp. 46-50.

[2] S. Wolfram, *The Mathematica Book*, 5th Edition, Wolfram Media, 2003.

[3] S. Pemmaraju and S. Skiena, *Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*, Cambridge University Press, 2003.

[4] B. Kaneva and D. Thiébaut, “Sorting algorithms,” Smith College, http://maven.smith.edu/~thiebaut/java/sort/demo.html.

[5] A. Jacobsen, “Data-link network protocol simulation,” University of Birmingham, School of Computer Science, http://www.cs.bham.ac.uk/~gkt/Teaching/SEM335/dlsim/Simulation.html.

[6] B. W. Watson, “The design and implementation of the FIRE engine: a C++ toolkit for finite automata and regular expressions,” *Computing Science Note 94/22*, Eindhoven University of Technology, Netherlands, 1994.

[7] Grail+ Project homepage, http://www.csd.uwo.ca/Research/grail/index.html.

[8] S. H. Rodger and T. W. Finley, *JFLAP: An Interactive Formal Languages and Automata Package*, Jones & Bartlett Publishers, Sudbury, MA, 2006.

[9] I. Skuliber, S. Srbljic, and I. Crkvenac, “Using interactivity in computer-facilitated learning for efficient comprehension of mathematical abstractions,” *Proceedings of the EUROCON 2001*, International Conference on Trends in Communication, Bratislava, Slovak Republic, July, 2001, vol. 2/2, pp. 278-281.

[10] I. Skuliber, S. Srbljic, and A. Milanovic, “Extending the textbook: a distributed tool for learning automata theory fundamentals,” *Proceedings of the 9th IEEE International Conference on Electronics, Circuits and Systems* (*ICECS 2002*), Dubrovnik, Croatia, September, 2002.

[11] E. Gansner, E. Koutsofios, and S. North, “Drawing graphs with *dot*”, January, 2006. http://www.graphviz.org/Documentation/dotguide.pdf

[12] E. R. Gansner and S. C. North, “An open graph visualization system and its application to software engineering”, *Software – Practice and Experience*, vol. 30, no. 11, November 2000, pp. 1203-1233.