Training Report
Realisation of a tool for manipulating and teaching automata
Université Joseph Fourier
Lab: Verimag
Supervisor: Yliès Falcone

Raphaël Jakse

August 2013

Contents

Introduction

In this internship, we developed a tool with the aim of easing teaching, learning and manipulation of automata. As for students, this tool can facilitate their learning by adding an interactive aspect to the lesson. This includes drawing automata, determinizing / minimizing them and more generally applying any algorithm of the lesson on them, run a word on it and write custom algorithms and test them on drawn automata.

1 Features

1.1 Drawing automata

Writing code being boring and off-putting for many people, like others tools of the domain, the tool provides a means to input automata by drawing them with the mouse and the keyboard in a hopefully intuitive way.

However, for repetitive tasks or big automata, it may be more convenient to write automata instead of drawing it. As a consequence, the tool also provides a way to input automata by writing them using a language which was maid as concise as possible.

In order to display results of algorithms or automata which were written rather than drawn, the tool embeds a Javascript version of Graphviz to render graphical version of automata automatically.

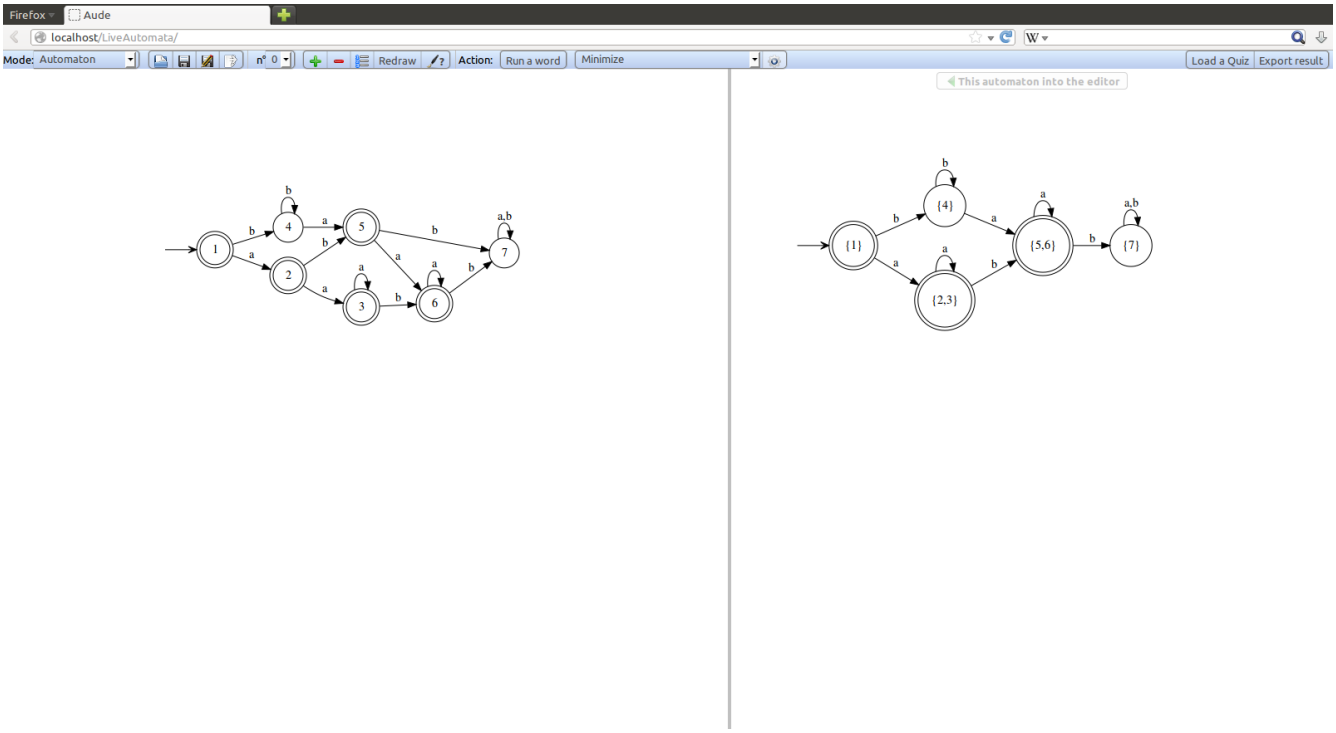
To help writing documents about automata, the tool provides a way to export images or DOT codes of automata produced by the user or the tool. DOT documents can then be translated into TiKZ format in order to include automata in a LaTeX document.

1.2 Algorithms

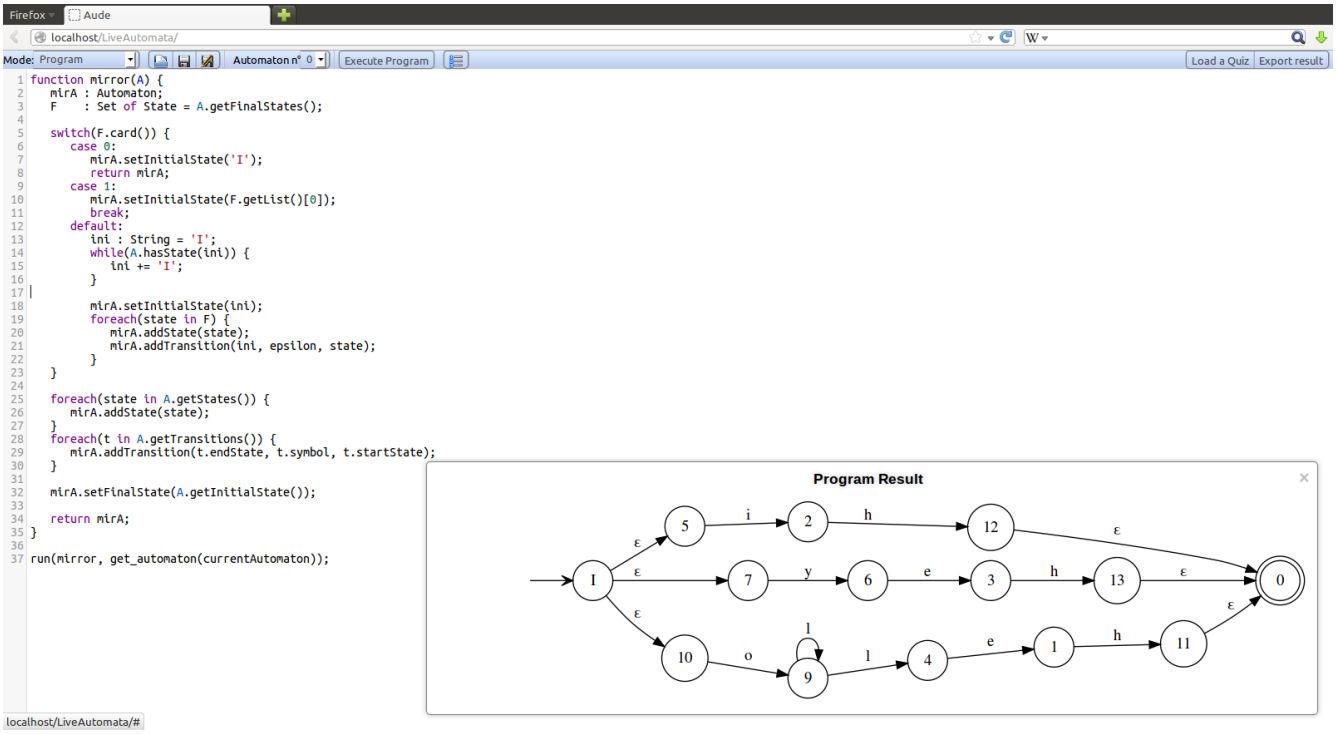
One of the most important features of the tool is its ability to run algorithms on automaton, and having these algorithm written and readable by the user in a dedicated language especially designed for manipulating sets and automata, still remaining generalist by being based on Javascript.

The tool comes with a tiny number of basic algorithms such as:

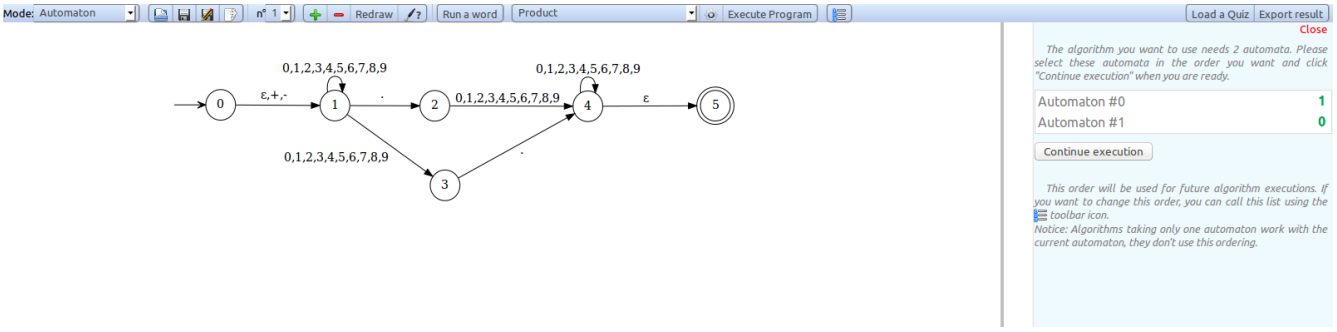
- Determinizaion
- Completion
- Minimization
- Epsilon removal
- Product
- Equivalence
- Complementation
- Empty and infinite language tests
- Regular expression to automaton



In addition to that, users can write their own algorithms using the same dedicated language as the one used for writing embedded algorithms.

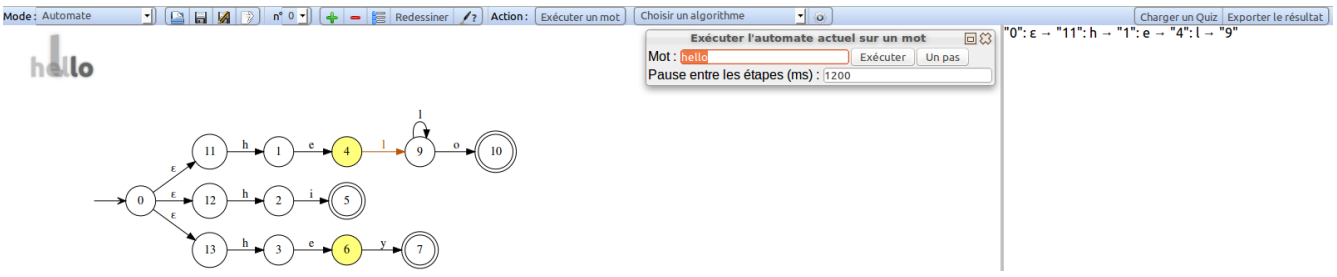


The tool can handle as many automata as necessary. That is to say, one can write an algorithm taking n automata, with n being any natural number (in the limits of the computer's resources). The user will be asked to choose which automata are to send to the algorithm when running it.



1.3 Running words

The tool features word execution on automata. Word execution is not an exclusive feature of this program, others tools can do this. However, special care was taken to make it pleasant and easy to follow by showing the execution progressively taking place, the word progressively being "eaten", with smooth transitions.



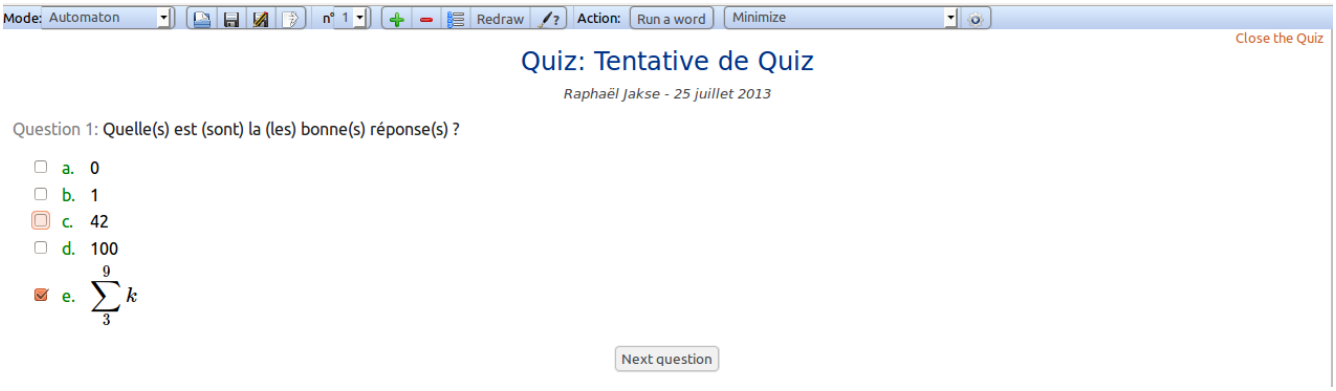
1.4 Quiz

With the tool, you can run custom quiz. The ability to run quizzes written by anybody (a teacher, a student) directly inside the tool, however, is probably an exclusive feature as of September 2013.

You can write or run quizzes featuring:

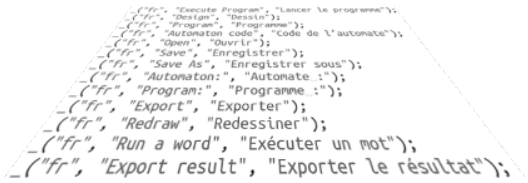
- Mere multiple choice questions, with zero, one or more answers, with any number of possible answers (stay reasonable however!)
- Questions that ask the user to draw an automaton corresponding to a set of words
- Questions that ask the user to draw an automaton corresponding to a language defined by an automaton or a regular expression in the quiz file.

LaTeX can be used to write mathematics in the quiz, thanks to MathJax.



1.5 Internationalization

It is almost always easier to learn something non-trivial in one's native language. While the tool is still only available in English and in French, it was conceived to make internationalization as easy as possible. Some work is still needed to make this happen, but not so much: in addition to a couple of lines of code, what is essentially needed is contributions from people who can speak well English and another language in which the tool is still not translated.



2 Technical choices

Several technical choices had to be taken to write the tool.

2.1 Technologies

Constraints were essentially:

- *efficiency*. The language and libraries used to develop the tool had to run fast and to allow fast development.
- *Web-compatible*. The idea behind being web-compatible is that users don't have to install anything to try and use the tool. Ideally, they just click on a link and they are instantly in front of the tool, with nothing to install but a browser.
- *server-independent*. While an online version of the tool could have extra features, it had to remain available for offline usage in order to be able to use the tool in any situation a computer could be used. Staying server-independent also ensures that the tool will scale well as the user base grows.

These constraints naturally led to the usage of HTML, CSS combined with Javascript, a reasonably fast, pleasant and widespread client-side language that doesn't need any special installation on the user's computer to run. Many people work with Javascript, this leads to the availability of tools of great quality around this language (debuggers, editors, engines, JIT compilers, browsers) and of many great libraries. Most notably, the existence of ports of Graphviz in Javascript is what made developing the tool as it is today in Javascript possible.

With regard to the language used for writing algorithms, we could have used Javascript directly, but Javascript is not well adapted for sets and therefore automata manipulation. The goal was to be able to write algorithms in a language which is close to their descriptions (e.g. in pseudo code), which use sets. We could have designed an entirely new language to match our expectations but that is counter-productive: it means writing a entire interpreter or compiler, which would have taken too much time and would have surely had bad performances. It would have led to an incomplete programming language to grow.

We instead opted for extending Javascript (as we already have what we need to run Javascript in a browser) with what we need: sets and some other constructions to make automata-related algorithms look better.

2.2 Extending Javascript

In our case, extending a programming language means:

- Extending the "standard library": adding a class to manipulate sets and a class to manipulate automata.
- Modifying the grammar of the language: adding features like set manipulations and iteration.

To do this, we basically need a function which takes a file written in our programming language which gives the corresponding pure Javascript code, with the following constraints:

- n^{th} line of the generated code must correspond to the n^{th} line of the input code, for accuracy in error reporting
- The generated code must be identical to the input code if the input code is pure Javascript.

To transform the input code into pure Javascript, regular expressions come in mind, as transformations seem quite simple:

Input code	**Generated code (simplified)**
`foreach(i in object) { ... }`	`object.forEach(function(i){...})`
`varname : Type`	`var varname = new Type`

However, this would be naive: This transformations must not be done inside string literals, *object* in foreach can contain parenthesis, the variable declaration can hold a initialization value, set literals look very similar to blocks of codes, foreach can be nested so a need to match curly brackets appears, break and return statement must be transformed inside foreach loop, etc. Manipulated languages are not *regular*[1] and transformations are not that trivial eventually. As a consequence, another method needs to be used, the code must be analyzed more subtly.

It turned out that the needed flexibility for extending the language comfortably required to recognize each instruction and expression recursively, which rapidly leads to reading corresponding parts of the ECMAScript documentation in order to know how to do it. Special cases and implicit semicolons (which are permitted in Javascript) need to be handled, which complicates things a bit.

A project like Jison, which is a parser like Bison might have been used[2]. However, handwriting the parser was chosen in order to keep whitespace characters intact, to have full control over optional semicolons (if a semicolon was not written by the programmer, the semicolon should not appear in the generated code) and to handle ambiguities between regular expression tokens (which begin with /) and division operators (/, /=) easily. Handwriting the parser also seemed to be a more efficient and straightforward solution here because transformations can be made without generating any abstract syntax tree. Moreover, this let write a quite flexible parser, validation being delegated to the actual Javascript engine, which already has a good error reporting system.

3 Comparison with others automata tools

Writing a tool for automata manipulation without looking to already existent projects would be a mistake: a tool with the same goals and the same expectations could already exist, making the creation of a new tool from scratch completely pointless. Contributing to such a project instead of beginning a new one would probably be a better approach.

As a consequence, it is essential to position the project among the others tools of the domain and tell what is new.

Like most tools of the domain, the tool gives the ability to the user to draw automata. However, unlike its friends, the user has more freedom in *choosing the shape of the transitions*, though Visual Automata Simulator is great for this, and thanks to the tight integration of Graphviz, automata can be *(re)drawn automatically* and have a *familiar look*.

Like most others tools, the program comes with basic common algorithms related to automata. However, it comes with *far more algorithms* than its friends and *lets the user write its own algorithms easily*, providing a language close to Javascript with sets as first-class citizens, which makes it suitable for manipulation of automata.

The program is designed with the user in mind: everything is thought to be the more pleasant and natural possible, appearance not being set aside. Ugliness and unfriendliness are bugs. An example of this is the graphical execution of a word: current states are seen in yellow, transitions being taken right now are brown and current final states are green. If a word runs out of the automaton, its states are drawn in red. The execution can be made *step by step* or not, and animations are designed to be beautiful and to ease the visualization of the execution.

What also make the program stand out is the Quiz Feature: it gives the ability to teachers and students to *write quiz for students* and these quizzes are run by the tool, using its automata manipulation capabilities. Questions of the quiz can be mere multiple choices questions as well as asking the user to write automata or regular expressions.

Another thing that can be said is that unlike others tools, this one is written with web technologies, which makes the program usable without any installation and will make the port on tablets easy. Thanks to web technologies, the program should work on any desktop operating system, provided a recent browser is installed, and the support for mobile operating systems should follow quickly.

[1] a language is regular iff it can be described with a regular expression. Rules like "there is the same number of opening and closing parenthesis" make a language not regular.

[2] See http://cjihrig.com/blog/creating-a-javascript-parser/ for an implementation of a Javascript parser with Jison

4 Conclusion about the internship

Though it was eventually not possible to use the program in concrete applications like real robots, and to do some research during the internship due to a lack of time, which is sad, I really enjoyed it.

What was great in this internship was to reach the computer science research community, to discover what a computer science lab looks like and how people of the lab work. I also really enjoyed working on a project I started myself months before and defining a great part of the internship subject with my supervisor.

This internship strengthened what I felt: research and how people work in this world is probably what suits me the best and it increased my confidence on my ability to work even if no explicit constraint is imposed. Working alone and together at the same time to discover new and build great things is also awesome.

Another great thing is I know that my program will be used to teach automata and this is amazing. I hope it will make automata learning more enjoyable or, at least, easier.

5 Links

- JFLAP: http://www.jflap.org/
- Visual Automata Simulator: http://www.cs.usfca.edu/~jbovet/vas.html
- jFAST - the Finite Automata Simulator: http://jfast-fsm-sim.sourceforge.net/
- Université Joseph Fourier: http://ujf-grenoble.fr/
- Verimag: http://www-verimag.imag.fr/