Automata Serialization for Manipulation and Drawing*

Miguel Ferreira[1], Nelma Moreira[2], and Rogério Reis[3]

1 CMUP & DCC, Faculdade de Ciências da Universidade do Porto, Porto, Portugal
miguelferreira108@gmail.com
2 CMUP & DCC, Faculdade de Ciências da Universidade do Porto, Porto, Portugal
nam@dcc.fc.up.pt
3 CMUP & DCC, Faculdade de Ciências da Universidade do Porto, Porto, Portugal
rvr@dcc.fc.up.pt

Abstract

GUItar is a GPL-licensed, cross-platform, graphical user interface for automata drawing and manipulation, written in C++ and Qt5. This tool offers support for styling, automatic layouts, several format exports and interface with any foreign finite automata manipulation library that can parse the serialized XML or JSON produced. In this paper we describe a new redesign of the GUItar framework and specially the method used to interface GUItar with automata manipulation libraries.

1998 ACM Subject Classification D.2.2 State diagrams

Keywords and phrases automata, serialization, visualization

Digital Object Identifier 10.4230/OASIcs.SLATE.2016.15

1 Introduction

Software environments for symbolic manipulation of formal languages and models of computation are widely recognized as important tools for theoretical and practical research, as well as pedagogical tools for teaching automata theory and formal languages. Examples include Grail+ [18, 12], OpenFST [13], FAdo [6, 1] and Vaucanson [11, 10]. The visualisation and interactive drawing of the diagrams of the computational models is also an important component, but few tools are available. Namely, JFLAP [15, 14] is mainly used for pedagogical purposes (and includes its own symbolic manipulator) and other alternatives include the use of generic graph visualization tools such as Graphviz [17].

The GUItar project aims to develop an extensible graphical environment for several combinatorial objects and models of computation, such as finite automata, pushdown automata, transducers, Turing machines, etc. Its functionalities include visualisation and interactive editing, i.e. automatic and assisted diagram drawing; and export/import filters that allow the interaction with several symbolic manipulators. Comparing with generic graph visualization tools several requirements are distinct and are analised in the following.

* This work was partially supported by CMUP (UID/MAT/00144/2013), which is funded by FCT (Portugal) with national (MEC) and european structural funds through the programs FEDER, under the partnership agreement PT2020.

Automatic graph drawing has been a very active research area and several (mainly) commercial software packages are now available for general and specific applications (data base design, information systems, bioinformatics, social networks,etc). In contrast, automata diagrams (alike labelled multi-digraphs) require additional aesthetics and graphical constraints: left-to-right reading, initial states on the left and final states on the right, edge shapes and label placements, etc. Another important issue is the visualisation of only some parts of a larger automata.

For the interactive editing it should define constraints that correspond to boolean functions of manipulators. For instance, if we are editing a deterministic finite automaton (DFA) no multiple transitions with the same label should be allowed; or a state can only be deleted if the resulting recognised language is the same.

For the interaction with the different symbolic manipulators (filters), it should be allowed the dynamic definition of actions that can be invoked, as well as conversions between the objects of the graphical framework and ones of the manipulators.

A first version of the software tool GUItar, was developed together with the FAdo system [1, 2, 3]. That version was implemented in wxPython [16] and included the visualisation and interactive editing for various types of automata. FAdo is mainly implemented in Python and currently includes most standard operations for the manipulation of regular languages and regular transductions, as well as several uniform random generators for these objects.

In this paper we present a new redesign of the GUItar framework that allows the interaction with several automata manipulators. In Section 2 we describe the main GUItar features within its interface and the several algorithms we use to layout the manipulated automata. The communication process between a library and GUItar is shown in Section 3. For the communication between these two layers to be possible, we needed to serialize the automaton. This is accomplished by the XML/JSON grammar presented in Section 4, along with the description of style manipulation of the automaton and examples of conversion to portable export formats. Section 5 concludes with some future work.

2 Graphical Interface for Automata Manipulation

The GUItar [8] is a graphical interface for automata drawing and manipulation, written in C++ and Qt5 [5]. The program is licensed with the GNU General Public License version 3.

It includes functionalities such as:

- Interactive "point and click" creation of automata;
- XML and JSON description of the automaton structure, styling and geometry of the states and transitions;
- Socket communication for obtaining and drawing automata represented on the GUItar XML or JSON language;
- Layout algorithms for state positioning;
- Embedded ipython shell for real time command line interaction;
- Exporting for several formats: PDF, PGF/Tikz, GraphViz (dot).

Its main window is composed of a QGraphicsScene widget used as a canvas, a terminal for interfacing with the scene automaton through command lines and selection buttons: insert state, insert transition and select items.

Using the same mechanism for automata manipulation through serialization described in Figure 1, GUItar supports file saving and opening, restoring all automaton style and geometry properties.

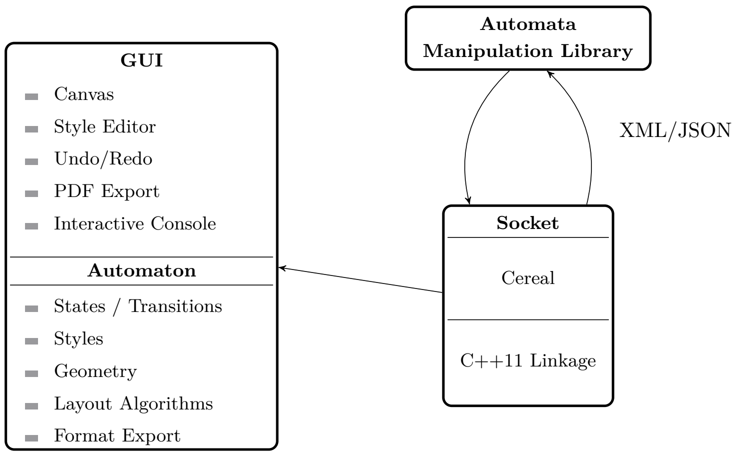


Figure 1 GUItar organization.

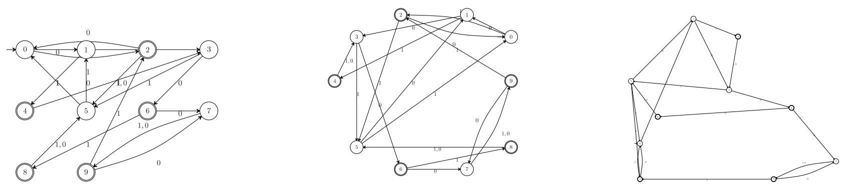


Figure 2 GUItar square, circle and spring layout of an automaton.

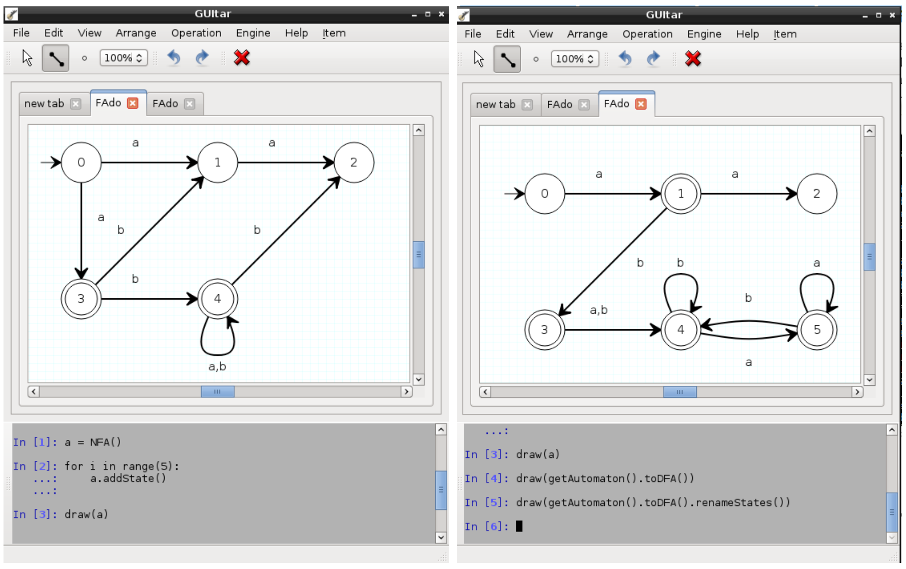
Comparing with the previous version of GUItar, the whole interaction and communication use a new paradigm (including the embedded shell), consisting on the complete separation of the symbolic manipulation program and the display program. Additional layouts and export formats where added.

2.1 Layouts

GUItar includes several implementations of graph layout algorithms, such as:

- Square - states are distributed in a $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$ virtual grid where n is the number of states.
- Circle - we calculate a circle radius as a function of the number of states and then each state s_i gets positioned at $radius \times (\cos(\Theta \times i), \sin(\Theta \times i))$ where Θ is $\frac{2\pi}{n}$ and n is the number of states.
- Two circle - based on the JFLAP implementation, we separate the automaton into two parts: the inner circle, which includes all states with degree greater than 2, and the outer circle which includes the remaining states.
- Barycenter [4] - an iterative algorithm that given a fixed set of states, the remaining states move towards the barycenter proportionally to its degree.
- Spring energy - another iterative algorithm that simulates a force system where states repel each other and transitions attract their states, working as springs. This algorithm can run a user-defined number of iterations or until the particle system's kinetic energy is below a certain threshold.

An example of some of these layouts is presented in Figure 2.


GUItar
File Edit View Arrange Operation Engine Help Item
100%
new tab
FAdo
FAdo
GUItar
File Edit View Arrange Operation Engine Help Item
100%
new tab
FAdo
FAdo

```
In [1]: a = NFA()

In [2]: for i in range(5):
   ...:     a.addState()
   ...:

In [3]: draw(a)
```

```
   ...:
In [3]: draw(a)

In [4]: draw(getAutomaton().toDFA())

In [5]: draw(getAutomaton().toDFA().renameStates())

In [6]: ▮
```

Figure 3 Example manipulation of an automaton using GUItar and FAdo.

3 Interfacing with Automata Manipulation Libraries

Communication with the GUItar is made through local sockets. For the sake of simplicity on the automata manipulation library side, we added support for the communication through the GUItar binary itself, as this is a single instance application.

There is also support for communication with a command line embedded inside the GUItar. This terminal has an ipython shell with the FAdo library loaded for a more natural way of handling the visible automaton.

The action for obtaining the currently seen automaton consists of sending the string "GET" through the GUItar socket and expecting a JSON/XML response. It is possible to draw using an analogous approach with the string "PUT" followed by the JSON/XML representation of the automaton, expecting empty answer in case of success and validation of the input against a schema.

This simplistic approach allowed us to quickly interface with known libraries such as FAdo and Vaucanson, by writing library-side methods for interpreting the JSON/XML GUItar format. An example using FAdo is shown in Figure 3.

4 Automata Serialization and its XML Grammar

GUItar uses the cereal [7] serialization library to produce JSON and XML representations of its automata to be understood by manipulation libraries. We chose this approach instead of other alternatives based on other serialization methods not only for the sake of code simplicity and parsing efficiency, but to allow any automata manipulation library programmer to easily interact with GUItar, be it through JSON, XML or even binary form.

The C++11 library cereal allowed us to transform our structural representation into a XML/JSON language that later can be parsed and validated using a schema. A fragment of the language schema is shown in Listing 2.

The structure is completely passed back and forth between the GUItar and the library through sockets and it is the library responsibility to maintain any state, if necessary, within

Listing 1 Partial JSON serialization example for the GUItar automata class.

```
{
 "automaton": {
  "title": "Automaton 1",
  "type": "",
  "states": [
   {
    "name": "1461439542",
    "label": "s2",
    "output": "",
    "initial": false,
    "final": true
   },
  ],
  "trans": [
   {
    "name": "14614395790",
    "orig_name": "1461439542",
    "dest_name": "1461439542",
    "label": "a",
    "weight": "",
    "compounds": []
   }
  ],
 "alphabet": ["a"]
 }
}
```

Listing 2 XML Relax NG Compact grammar for the GUItar automata class.

```
start = element cereal {
 element automaton {
  element title { text },
  element type { text },
  element states {
   element state {
    element name { xsd:integer },
    element label { text },
    element output { text },
    element initial { xsd:boolean },
    element final { xsd:boolean }
   }*
  },
  element trans {
   element transition {
    element name { xsd:integer },
    element orig_name { xsd:integer },
    element dest_name { xsd:integer },
    element label { text },
    element weight { text },
    element compounds {
     element compound {
      element key { text },
      element value {text}
     }*
    }
   }*
  },
  element alphabet {
    element symbol {text}*
  }
}
```

the extra branch of the grammar. This branch is guaranteed by GUItar to be returned exactly as it was sent.

For different automaton types, we allow special transitions with compounds, in which a format string is defined on the label field in the form of $\$compound_1 \cdots \$compound_n$, and the values of the compounds are stored as key-value pairs on its branch.

4.1 Example of a XML Automaton with Styles

On this implementation of GUItar, we include support for styling inside the JSON/XML grammar. Each drawable object can have style properties in GUItar defined for each object class. There can be style templates defined inside the GUI besides the default one.

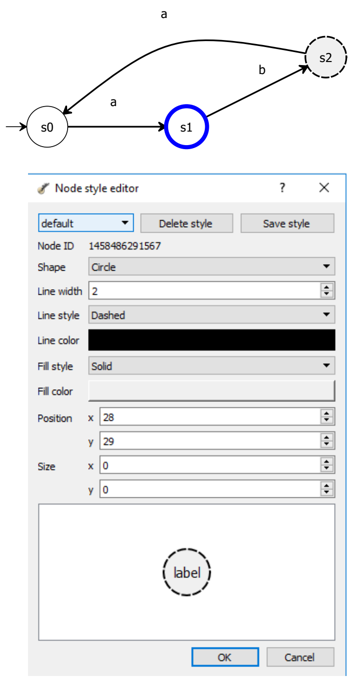
The correspondence between style XML, automaton visualization and style form can be seen in Listing 3 and Figure 4.

4.2 Exporting to Visualization Formats

Our automata objects in GUItar can be exported to several known formats. At the moment, it is possible to export to Vaucanson-G [9] and PGF/Tikz maintaining some level of similarity between geometry and style. We can also export the automaton with some level of styling to the GraphViz layout program in dot language.

Listing 3 An example of XML GUItar.

```
<?xml version="1.0" encoding="utf-8"?>
<cereal>
 <automaton>
  <title>FAdo</title>
  ...
  <states size="dynamic"> ... </states>
  <trans size="dynamic"> ... </trans>
 </automaton>
 ...
 <style>
  <states size="dynamic">
   <value0>
    <key>1458486283455</key>
    <value>
     <shape>0</shape>
     <lineWidth>5</lineWidth>
     <lineStyle>0</lineStyle>
     <lineRgb>255</lineRgb>
     <fillStyle>0</fillStyle>
     <fillRgb>16777215</fillRgb>
    </value>
   </value0>
  </states>
 </style>
</automaton>
```



Node style editor

default | Delete style | Save style

Node ID 1458486291567
Shape: Circle
Line width: 2
Line style: Dashed
Line color
Fill style: Solid
Fill color
Position x: 28
y: 29
Size x: 0
y: 0

Figure 4 Automaton state styling.

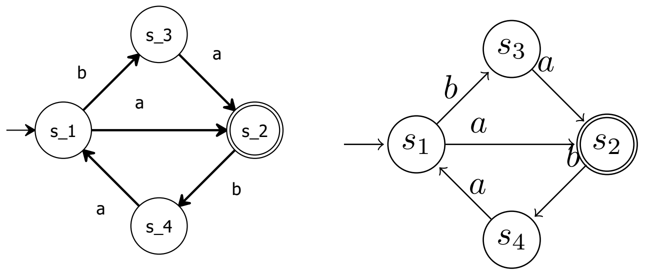


Figure 5 GUItar native export and PGF/Tikz comparison.

It is possible to export to a PDF file, maintaining an exact layout, the visible drawing in GUItar through Qt framework methods.

5 Conclusion

In this paper we presented a new implementation of the GUItar program for visually manipulating automata and interacting with libraries. This new version, although using some ideas of the previous one, consisted in a new redesign of most of the features and addition of new ones. Major improvement was the possibility of communicate with several automata symbolic manipulators.

This project is still under continuous development. The simplistic socket interface combined with the serialization procedure provides an almost transparent communication with automata manipulation libraries.

There are still features to develop and add to this project, such as GraphML export format, more layout algorithms focused on automata drawing and constrained edition driven by manipulators functions.

Acknowledgements. We want to thank to the anonymous reviewers for their comments that helped to improve this paper.

References

1 André Almeida, Marco Almeida, José Alves, Nelma Moreira, and Rogério Reis. FAdo and GUItar: tools for automata manipulation and visualization. In Sebastian Maneth, editor, *14th International Conference on Implementation and Application of Automata, CIAA 2009. Proceedings*, volume 5642, pages 65–74, Sidney, July 2009. Springer.

2 André Almeida, Nelma Moreira, and Rogério Reis. GUItar and FAgoo: Graphical interface for automata visualization, editing, and interaction. In Luís S. Barbosa and Miguel P. Correia, editors, *Inforum, Simpósio de Informática*, pages 317–328, Braga,Portugal, 9-10 Setembro 2010.

3 José Alves, Nelma Moreira, and Rogério Reis. XML description for automata manipulations. In Alberto Simões, Daniela Cruz, and José Carlos Ramalho, editors, *Actas XATA 2010, XML: aplicações e tecnologias associadas*, pages 77–88, ESEIG, Vila do Conde, 2010.

4 Giuseppe Di Battista. *Graph drawing: algorithms for the visualization of graphs.* Pretice Hall, 1999.

5 The Qt Company. Qt, Access date:1.12.2015. URL: `http://www.qt.io`.

6 Project FAdo. FAdo: tools for formal languages manipulation, Access date:1.11.2015. URL: `http://fado.dcc.fc.up.pt/`.

7 Shane Grant and Randolph Voorhies. cereal – A C++11 library for serialization, Access date:4.14.2016. URL: `http://uscilab.github.io/cereal/`.

8 Project GUItar. GUItar, Access date:1.06.2016. URL: `http://guitar.dcc.fc.up.pt/`.

9 S. Lombardy and J. Sakarovitch. Vaucanson-G, Access date:1.12.2015. URL: `http://igm.univ-mlv.fr/~lombardy/Vaucanson-G/`.

10 Sylvain Lombardy, Yann Régis-Gianas, and Jacques Sakarovitch. Introducing Vaucanson. *Theor. Comput. Sci.*, 328(1-2):77–96, 2004. `doi:http://dx.doi.org/10.1016/j.tcs.2004.07.007`.

11 Sylvain Lombardy and Jacques Sakarovitch. Vaucanson, Access date:1.12.2015. URL: `http://vaucanson-project.org`.

12 Darrell Raymond and Derick Wood. Grail: A C++ Library for automata and expressions. *J. Symb. Comp.*, 17(4):341–350, 1994.

13 Michael Riley. OpenFst, Access date:1.3.2016. URL: `http://www.openfst.org`.

14 Susan Rodger and Thomas Finley. *JFLAP: An Interactive Formal Languages and Automata Package.* Jones and Bartlett Publishers, 2006.

15 Susan H. Rodger. JFLAP, Access date:1.12.2015. URL: `http://www.jflap.org`.

16 Julian Smart, Robert Roebling, Vadim Zeitlin, and Robin Dunn. *wxWidgets 2.6.3: A portable C++ and Python GUI toolkit*, 2006. URL: `http://wxpython.org`.

17 Graph Visualization Software. Graphviz, Access date:1.12.2015. URL: `http://graphviz.org/`.

18 Sheng Yu and Cezar Campeanu. Grail+, Access date:1.3.2016. URL: `http://www.csit.upei.ca/~ccampeanu/Grail`.