Automata Lab: An Open-Source Automata Visualization, Simulation, and Manipulation Tool

D. Kramp [1]***, J. Wadden***[1]***, K. Skadron***[1]

We present Automata Lab, an open-source homogeneous automata visualization, manipulation, and simulation tool. Existing automata visualization tools are closed-source, slow, and/or restricted to non-homogeneous automata. With the advent of spatial hardware accelerators for homogeneous finite automata, it is desirable to have a fully open-source, flexibly licensed visualization tool to help educate developers, develop and debug new automata applications, and gain insight into how to best accelerate automata processing. Automata Lab is browser-based, and allows access from any browser-enabled device (even mobile devices) once installed on a host system.

Keywords: automata processing, non-deterministic finite automata, parallel processing, simulation, benchmarking.

1. Introduction

Recent development of spatial accelerators for automata processing [3] have spurred a large amount of research investigating new and non-obvious use-cases for automata processing [7]. Some application kernels such as approximate pattern search, are straight forward to represent as regular expressions. However, other applications are difficult or non-intuitive to represent as regular expressions. Thus, it is desirable to be able to develop and prototype automata-based applications directly as automata graphs.

Existing tools to design, prototype, and debug automata-based applications are lacking. JFLAP [6] is an automata visualization and manipulation tool. However, JFLAP does not allow restriction to homogeneous automata (required by many accelerators [3, 4]) and is not freely licensed, making industry adoption and development problematic.

To support application development for their Automata Processor, Micron has built the AP Workbench [1]. AP Workbench is a portable desktop application for visualization and development of automata-processor applications. AP Workbench supports homogeneous automata manipulation and simulation, but is not freely available, not open source, and limited in the size of the automata it can practically visualize and simulate. Furthermore, the AP Workbench does not include classic automata processing optimizations or transformations commonly available in optimizing automata compilers.

To solve these problems, we present Automata Lab. Automata Lab is firstly open-sourced and freely licensed, so that both industry and academia can use and contribute code for whatever purpose necessary. Automata Lab is build around the VASim [8] virtual automata library and is designed to handle visualization and simulation of graphs with hundreds of thousands of nodes. We hope that these qualities will allow Automata Lab to become the de-facto standard Automata Education tool and be integrated into both academic and industry tool chains.

2. Background

2.1. Automata Processing

Automata are directed graphs with transition rules between nodes. An input stream of symbols guides transitions between nodes. In this manner, automata can be used to recognize patterns in input text. Homogeneous automata are a special form of general automata that

[1]University of Virginia, Dept. of Computer Science: {dankramp,wadden,skadron}virginia.edu

restrict all incoming transitions into a state to have the same rule. Homogeneous automata (also known as Glushkov automata), are desireable because they are easy to simulate in software, and also easy to represent as a circuit in reconfigurable hardware.

Because automata processing may require many parallel, unpredictable transition rule look-ups, (commonly known as "pointer chasing") automata processing is challenging to accelerate on von Neumann computer architectures [7]. However, spatial architectures are ideal automata processors. Spatial architectures (such as FPGAs), can "lay out" homogeneous automata circuits in a reconfigurable fabric. Just as FPGAs can efficiently emulate boolean circuits in parallel, spatial automata processing architectures can efficiently emulate automata circuits in parallel. All homogeneous automata states can consider inputs, transition rule computation, and propagate outputs within a single cycle, while von Neumann architectures may take many hundreds or thousands of cycles.

2.2. JFLAP

JFLAP [6] is an open source, object-oriented automata processing library. JFLAP has a graphical user interface that allows for design, debug, and experimentation of automata. JFLAP also includes many well-known automata-related algorithms, such as conversion from regular expressions to automata, conversions from non-deterministic finite automata to deterministic finite automata, etc... While JFLAP is useful for education purposes, its license precludes use in software sold for profit. This restricts industry participation, and is an undesirable property .

2.3. AP Workbench

AP Workbench is a closed-source tool distributed by Micron [1]. AP Workbench can visualize ANML files and allows users to manipulate and simulate ANML automata graphs. Because Micron's Automata Processor and ANML language support special boolean and counter elements [3], AP Workbench also supports visualization, creation, and simulation of these elements.

While AP Workbench is fully featured, it can be extremely slow for even moderately large automata graphs, and cannot load most ANMLZoo [7] benchmarks because they are too large. Furthermore, Micron's tool is not publicly available or freely licensed making gaining access to the tool difficult.

3. Automata Lab

This paper presents Automata Lab, an automata visualization, manipulation, and simulation tool. Automata Lab is freely licensed under BSD 3-clause for both education and industry use. It is open-source, allowing crowd-sourcing of bug-fixes, feature enhancements, and other code improvements. Automata Lab also uses the VASim automata processing library [8] as a powerful automata manipulation and simulation back end. The following sections highlight the most important features of Automata Lab and some success stories from internal usage of the tool.

3.1. Browser-based Graphical User Interface

Automata Lab is a browser-based tool. The software relies on the Wt SDK [5] for developing web applications in C++ and the Sigma.js javascript graph visualization library [2]. A browser-

based application has many benefits over a stand-alone app installed on a host system. Firstly, once the server is installed, anyone with access to a browser can use the simulator. In practice, this has been extremely important to quick user adoption within the Center for Automata Processing. This case study is discussed later in the paper. Secondly, browser-based applications can be run on a variety of devices. While the server code is not arbitrarily portable, once installed, users can access Automata Lab from any browser enabled device. Our preferred presentation device has become touch screen computers or tablets where users can easily manipulate the control panel with their hands. Even low-power devices such as smart phones are able to easily display extremely large graphs, while the server handles the burden of simulation.

A stand-alone application that runs entirely within a host desktop environment would most likely be faster. However, the above advantages far outweigh the added speed. Automata Lab can be locally hosted and used on the same machine, and so an internet connection is not required.

3.2. Visualization Mode

The default mode of Automata Lab is *visualization mode*. Visualization mode gives a view of the automata graph in the Visualization Pane and allows you to explore different portions by dragging the window and zooming in using the mouse wheel. If viewing on a mobile device, navigation is as simple as swiping with your fingers, and pinching to zoom. A figure showing visualization mode and the Visualization Pane is shown in Figure 1.

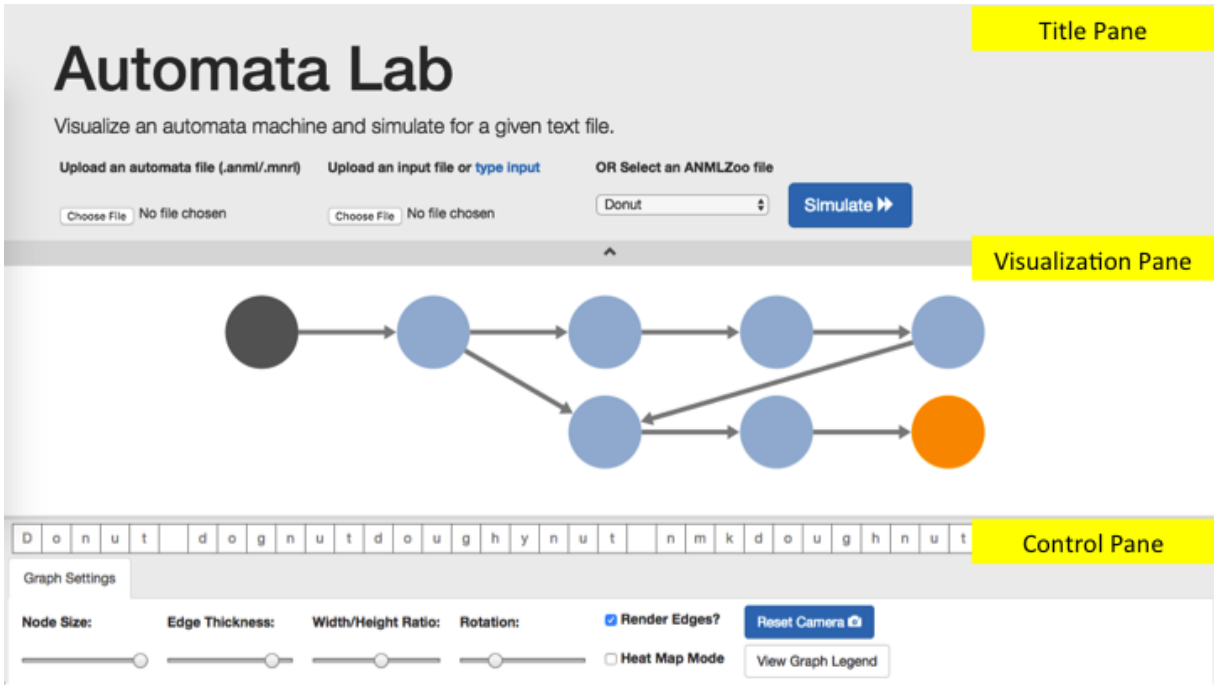

Figure 1. The abstract spatial automata processor system.

Automata can be loaded from files located on the individual client's computer, or from a set of stock automata graphs from the ANMLZoo benchmark suite using the drop down menu in the Title Pane.

Once loaded, some display properties of the automata are able to be controlled via slider bars in the *Graph Settings* tab in the *Control Pane* at the bottom of the screen. Node and edge

size can be changed independently of each other. The *Width/Height Ratio* slider controls the shape of the box each automata subgraph is displayed in. The *Rotation* slider rotates the graph in increments of 15 degrees. Automata rotation can also be accomplished on touch screens using two fingers.

The *Render Edges?* check box turns edge rendering on or off. Edge rendering can slow graph re-draw for some larger graphs, and can be turned off for faster graph exploration. The *Heat Map Mode* check box displays how active individual states are during simulation. This can be used to characterize automata behavior by identifying hot-spots.

Properties of individual automata can be inspected by right clicking on individual automata elements. A right click will display the element's ID, symbol set, start mode if it is a start state, and the report code if it is a reporting state.

Right clicking a state also gives the ability to display only the state's children, or hide the state's children and display the rest of the graph.

A variety of options are available when the visualization pane is right clicked. For example, automata can be searched by state ID strings by right clicking on the visualization pane, selecting *Search by ID*, and entering the string ID of the state.

3.3. Editor Mode

Automata lab is capable of editing automata in the Visualization Pane. To enter *editor mode*, the user can right click on the visualization pane and select *Toggle Editor Mode*. Once in editor mode, the background of the Visualization Pane will turn slightly yellow and an *Editor Tools* tab will appear in the Control Pane.

Inside editor mode, elements can be moved, deleted, or created. To move an element, the user can simply drag it with the mouse. Properties of an element can be changed by right clicking on the element and selecting *Change Data*. New connections can be added by selecting *Add Outgoing Edge* and selecting the state to which you would like to add a transition. Elements can be deleted by right clicking and selecting *Delete*. When deleted, all associated connections are also removed.

Groups of elements can be selected by using the *Lasso Tool* or *Box Selector* in the Editor Tools tab. Other shorcuts for element selection are also displayed in the Editor Tools tab. For instance, *spacebar* + e selects all neighboring elements of the selected element.

After the user has edited the automata, they have the option to export the new automata object as an ANML file and export the graph position data as a JSON file, which can be loaded along with the automata file in the future for custom positioning.

To exit Editor mode, the user can simply right click on the Visualization Pane and select *Toggle Editor Mode* a second time.

3.4. Simulation Mode

Simulation mode allows automata to be simulated using an input stream of symbols. There are two ways to specify an input stream for simulation. First, a user can upload an input file using the button in the title pane. Second, users can type their own input from the keyboard by clicking the "type input" link. Stock automata come with stock input streams, which are automatically loaded along with the automata graphs. Once an input stream is loaded, it is displayed just above the Control Pane in the bottom portion of the screen.

Simulation can be initiated by clicking the blue *Simulate* button in the Title Pane. Once clicked, a *Simulation Tools* tab should appear in the control pane. The Simulation Tools tab gives users control over simulation. Users can step forward or back or can automatically play simulation at varying speeds. Users can also set break points in execution using the *Stop on:* drop down menu. Break points can be set for particular cycles, or when a report is encountered.

The simulation functionality implements a cache system for quick traversal of cycles. One hundred cycles are loaded at a time, and up to 150 are accessible at any given time. The cycles within the current cache are displayed in the character stream as clickable text whereas characters outside the cache appear disabled and unclickable. As the user progresses through the simulation, the cache advances forward each time the user selects a cycle that is within 25 cycles of the cache's end.

During simulation, the current symbol being considered is highlighted in the character stream. If an input symbol causes a report, the cell turns magenta. Simulation is also visualized in the visualization pane by changing the colors of enabled, and activated elements in the automata. A yellow state indicates the state is enabled and attempts to match on the current input signal. If the element matches, it turns yellow, and propagates its signal to all children elements. A full color key can be viewed by clicking the *Graph Legend* button in the *Graph Settings* tab of the control pane.

3.5. Case Study Success Stories

The following two stories highlight the impact Automata Lab has already had while prototyped within the Center for Automata Processing. The first story shows how Automata Lab was able to quickly expose a bug in the VASim simulation algorithm that was not otherwise obvious. The second story shows how the browser-based nature of Automata Lab enabled quick adoption within the group, and enabled success of even the most inexperienced students.

3.5.1. VASim Bug Fixes

To be used as a simulation library, rather than a command line tool, the Virtual Automata Simulator library had to go through a massive refactoring. This refactoring led to lots of changes to the core simulation code that went unaudited. During this process, a bug was introduced to the simulation core that caused elements that were supposed to be activated only on the first cycle, to be activated on the first two cycles. This bug did not cause any changes in behavior in the ANMLZoo applications and so went unnoticed for weeks. Discrepancies in output first occurred when developing a new automata application. The student developing the application noticed incorrect behavior while watching simulation. The bug was easy to reproduce and diagnose given the visual "bug report" and immediately fixed. A bug that may have taken hours to diagnose took minutes due to the availability of the Automata Lab visualization tool.

3.5.2. New User Setup

During the Summer, the Center for Automata Processing hosted seven undergrad and high-school interns with limited to no experience with automata processing. Students with such limited experience can take weeks to learn the required skills necessary to begin research and development of an automata-based application. Visualization tools, such as AP Workbench, allow interns, even those who have no experience with automata processing, to quickly self-learn

the execution model, and explore new applications of accelerated automata processing. While AP workbench is a useful tool, it is difficult to license and install, and also limited to Linux and Windows operating systems.

Within a week of the release of the first functional prototype of Automata Lab, all interns and most members of CAP had switched from AP Workbench. Using the tool was as easy as visiting a web address on a shared server. Thus, users did not need to manage an installation or license. Not only was Automata Lab easy to use for new users, but it allowed functionality that led to many application debugging successes. Within a few weeks, most memebers of CAP were positively impacted by Automata Lab's release.

4. Conclusions and Future Work

This paper presented Automata Lab, an automata visualization, manipulation, and simulation tool. Automata Lab freely licensed allowing free usage in academia and industry. Automata Lab is also open-source, allowing crowd sourcing of feature extensions and bug fixes. Automata Lab has seen quick adoption within the Center for Automata Processing and we hope will become the de-facto tool for automata processing application design and debug.

Source code for Automata Lab can be found on GitHub via `https://www.github.com/dankramp/AutomataLab`. The Automata Lab GitHub page also has extensive documentation in its wiki.

5. Acknowledgements

This work is supported by the Center for Automata Processing and Achievement Awards for College Scientists (ARCS).

References

1. Micron Automata Processor SDK. `http://micronautomata.com/`.
2. Alexis Jacomy. Sigma.js: A JavaScript Library for Graph Drawing. `https://github.com/jacomyal/sigma.js`.
3. Paul Dlugosch, Dave Brown, Paul Glendenning, Michael Leventhal, and Harold Noyes. An efficient and scalable semiconductor architecture for parallel automata processing. *IEEE Transactions on Parallel and Distributed Systems*, 25(12):3088–3098, 2014.
4. Intel. Hyperscan. `https://github.com/01org/hyperscan`.
5. Koen Deforche. Wt: A C++ Web Toolkit. `https://github.com/emweb/wt`.
6. S.H. Rodger and T.W. Finley. *JFLAP: An Interactive Formal Languages and Automata Package*. Jones and Bartlett, 2006.
7. Jack Wadden, Vinh Dang, Nathan Brunelle, Tom Tracy II, Deyuan Guo, Elaheh Sadredini, Ke Wang, Chunkun Bo, Gabriel Robins, Mircea Stan, and Kevin Skadron. ANMLZoo: A Benchmark Suite for Exploring Bottlenecks in Automata Processing Engines and Architectures. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, 2017.
8. Jack Wadden and Kevin Skadron. VASim: An Open Virtual Automata Simulator for Automata Processing Application and Architecture Research. Technical Report CS2016-03, University of Virginia, 2016.