Linnæus University
Sweden

Degree project

GaniFA NG

The Next Generation of Algorithm Visualizations of Finite Automata

Author: Hüsamettin Yüksel
Supervisor: Prof. Dr. Andreas Kerren

Date: 2014-04-15

Course Code: 5DV00E, 30 credits
Level: Master

*D*epartment of Computer Science

Abstract

Software visualization is a promising field in computer science. Visualization of software-related data deals with aspects that are connected with the visual representation of computational data sets or simulations in order to achieve a deeper understanding or a simpler representation of a more complex phenomenon. Interactive software visualizations are also valuable active learning techniques that can improve explorative learning in computer science. Since dynamic processes such as the working of an algorithm or the flow of information between computing entities are not well explained by static media styles like text or images visualization has a substantial role in effective learning.

In this thesis, we will study on modernization of an educational software called GaniFA which was developed for the visualization of finite automata at the end of 1990s. The modernized software is called GaniFA NG (Next Generation) and enables designing finite automata and simulating all the functioning processes.

We will discuss benefits of visualization in automata theory learning and the software development life cycle of our proposed visualization tool in this report, and the experience derived from the implementation process will also be documented.

Keywords: Finite automata, education, learning, computer science education, visualization, software visualization, automata theory, JUNG.

Acknowledgements

First and foremost I offer my sincerest gratitude to my supervisor, Professor Dr. Andreas Kerren, who has supported me throughout my thesis with his patience and knowledge. I attribute the level of my Master's degree to his encouragement and effort, and without him this thesis, too, would not have been completed or written. One simply could not wish for a better or friendlier supervisor. Especially, I would like to give my special thanks to my lovely wife, Maide YÜKSEL, for her patient love and support that enabled me to complete this work.

List of Figures

List of Tables

Contents

1 Introduction

It is a well-known fact that it is difficult to understand abstract concepts in theoretical computer science. Almost all automata theory documents or textbooks present material in a former manner with little visualization and no direct feedback. This being the case, making the students to love the theoretical computer science is getting harder. Unlike a typical textbook, a software visualization tool that allows students to explore finite state automata topics in a visual and interactive manner has a remarkable occasion of increasing students' self-efficacy. Software visualization is a process of transforming information into a comprehensible visual form, enabling students to understand the matter easily.

This chapter is going to introduce the general background of the thesis and the motivation to choose this particular topic. Other parts are the goal of this work and restrictions.

1.1 Motivation and Aim to Work

Many students may find automaton generation from regular expression and conversion of non-deterministic finite state automata (NFAs) into deterministic finite state automata (DFAs) [17] difficult to understand. These processes are often inadequately presented in traditional textbooks as they cannot show the dynamic nature of the generation process. The aim of this thesis is providing a tool that will help students better understand the conversion, and may help teachers to better present the conversion. This tool is designed to be beneficial to student learning; therefore, many different learning modes were included. These modes vary the speed and level of detail of the conversion so that students may proceed at their own pace. They also have a chance to analyze their own examples. Each step of the processes is animated, and the student is constantly made aware of what has just happened and what is going to happen next, along with how the step will be performed. If the student is asked by the tool to supply anything, the tool will provide the student with feedback on his/her mistakes or successes.

1.2 Goal of This Thesis

Considering the conveniences and advantages of visualization, the goal was to develop a visual concept in which users can analyze finite state automata according to points of interest of them. Our tool, called GaniFA NG, was developed with the intention of benefit for both teachers while teaching automata theory and students during lectures or labs to work on examples or assignments. For that reason, it can be called educational visualization software. On the other hand, this software not only provides the results of operations, it also visualizes the processes of operations. Animations are used effectively for demonstrating the progression. This software is useful for those students who want to make more practice, to reproduce examples illustrated in lecture or to create their own examples. They can monitor their own input during visualization quickly and easily. GaniFA NG is aimed to be used in Formal Languages and Automata Theory lectures by students for the sake of their personal development. It is recommended to use visualization concepts of GaniFA NG for education of theoretical computer science. The declared goal can be enlarged upon item by item as follows:

- Converting a regular expression into a NFA that accepts the same language is easy with standard algorithms. Also the constitution of the automaton is suggestive for the user or learner. Our tool has an animated process of this conversion so that the users can examine both the formation process and the result automaton.

- Every NFA has an equivalent DFA, and there are several steps of the conversion method. These steps are sometimes easy and hard to follow. While a NFA is simpler in terms of the number of states, its rules are quite subtle. DFAs are more complex in terms of the number of nodes, but with simpler rules since there is exactly one output state for any state. Our software enables users to visualize this conversion.

- Minimizing NFA means removing the ϵ-transitions from the automaton while preserving the automaton nature. GaniFA NG aims at providing an illustrative and explicit animation for this minimization to the users with no data loss.

- Minimization of DFA is a bit different from the minimization of NFA since DFA has no ϵ-transitions, and the minimization means to remove some duplicated transitions. One of the aims of this tool is to ensure that the users are able to comprehend the steps of this minimization operation.

- The last, most important aim of this thesis is enabling users to visualize their input data with the given automaton (NFA or DFA). Repetitive sequence of input string compels the users to follow and check if the input is valid or not. GaniFA NG has visualization features for all of the automaton types: NFA, DFA, minimized NFA, or minimized DFA can be visualized and studied.

1.3 Restrictions

Since the main focus of the developed tool was analyzing the regular expressions, and visualizing corresponding finite state automata, an updated, but still plain tree-based layout is applied during the visualization processes of finite state automata graphs. JUNG Graph Framework underlies the graph structure of this tool. So, the layout organization is provided by it.

1.4 Structure of This Thesis Report

The whole report of the thesis work consists of 5 chapters. The first three chapters are about theoretical work, and followed by the implementation part which starts with the fourth chapter. Chapter 2 states the background knowledge of automata theory, learning theory, and finite state automata (FSA). The review of environment features, concepts, and capabilities of GaniFA NG are explained in Chapter 3. Chapter 4 is the implementation chapter in which some code samples, implementation process details, and the applied design patterns are presented. The fifth chapter concludes the thesis work and explains the future work plans.

2 Background

This section deals with the actual background of the work. A brief introduction into automata theory is given as well as learning theory. Animation in learning systems is discussed later on in Section 2.3 leading to visualization background.

2.1 Automata Theory

Automata theory is the study of the computational problems that can be solved using abstract computing devices or machines [26] and based on mathematical computations. These mathematical computations are used to symbolize mathematical models.

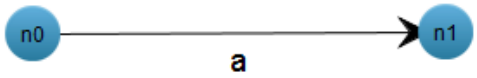
So, what is a mathematical model? A mathematical model, for example, a finite state machine, is equivalent to real computers and real programming languages. It means any problem that can be solved by a real computer can also be solved by using these models. Finite state machines store the status of something at a given time and can operate on input to change the status, and/or cause an action or output to take place for any given change [10].

Anything that holds information can be in various states. For instance, an on-off light switch has two states: on, off. A computer can be considered as a state machine and also has certain number of states. Each data register of a computer stores a state. The read-only memory from which a boot program is loaded stores a state (the boot program itself is an initial state). The operating system is itself a state, and each application that runs begins with some initial state that may change as it begins to handle input. Thus, at any moment in time, a computer system can be seen as a very complex set of states, and each program in it as a state machine. In practice, however, state machines are used to develop and describe specific device or program interactions.

A finite state machine or finite state automaton (plural automata) consists of a number of states (an infinite state machine can be deduced, but will not be practical), represented by circles in Figure 1(a), and transitions, represented by arrows in Figure 1(b). Every automaton has to start with a start state which is shown in Figure 1(c), and at least one final state shown in Figure 1(d). In other words every automaton has to have a start state. The final state and start state may be the same in some cases, and it is represented by a double surrounded circle shown in Figure 1(e). When the automaton considers a symbol of input, a transition occurs to another state according to its transition function (empty transition is allowed in NFA which is shown in Figure 1(f)). Transition function takes the current state and transition symbol as parameter and specifies the next state of the machine. Next state may be the current state (can be seen in Figure 1(g)) as it can be a different one. Transition symbol can be of any length. (Some rules restrict the length of transition parameter, but others allow any length at all.)

The finite state automaton in Figure 2.2 starts in state “n0”. To be accepted, an input must be “a”. Therefore, input “a” leads from state “n0” to “n1”. Any other input at state “n0” leads to a reject state. Only input “b” leads to the valid state (n2), and any other input is rejected. Once the machine is in state “n2” automaton actually reaches the final state. There are also two alternatives so that two arrows from state “n2”. If the input is “a”, it leads to state “n1” again. Input “b” means a self-loop edge from state “n2” to itself.

(a) A sample state (node) of a finite state machine

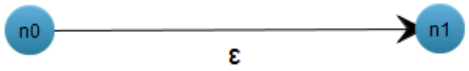


(b) A transition (edge) reads “a” as input symbol

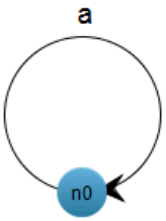
(c) A sample node with which automaton starts

(d) A node represents a valid input finish

(e) A node which represents both the start state and a valid final state



(f) An empty transition edge



(g) A self-looping edge

Figure 2.1: Elements of a finite state automaton

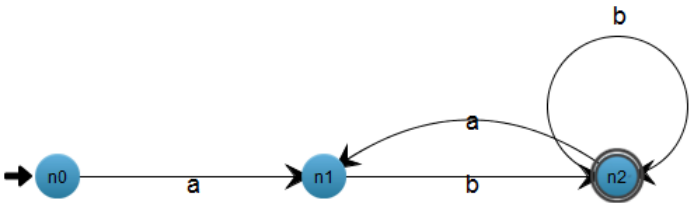


Figure 2.2: A sample finite state machine

To summarize, a finite state automaton can be described as:

- An initial state of something stored some place.
- A set of possible input events.
- A set of new states that may result from the input.
- A set of possible output events that result from a new state.

An automaton with two arrows from the same state for the same input is called a Non-deterministic Finite Automaton (NFA). If every state has exactly one transition for each possible input, it is called Deterministic Finite Automaton (DFA). NFAs are more complicated than DFA. But DFAs are just powerful as NFA. Figure 2.3 and Figure 2.4 are example of each.

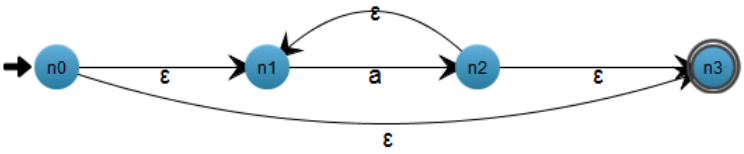


Figure 2.3: Non-deterministic finite automaton (NFA) of REG a*

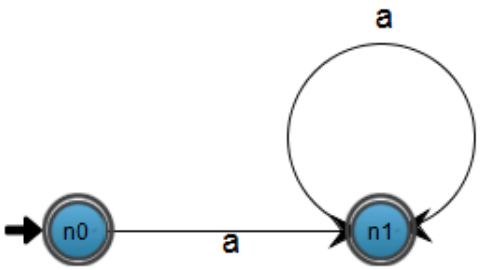


Figure 2.4: Corresponding deterministic finite automaton (DFA) of REG a*

An NFA is a mathematical model which is represented formally by a 5-tuple, (Q, Σ, Δ, q0, F), consists of [20]:

- A finite set of states Q.
- A finite set of input (transition) symbols Σ.
- A transition relation $\Delta \subseteq Q \times \Sigma \times Q$.
- A start state $q0 \in Q$.
- A set of final (or accepting) states $F \subseteq Q$.

An NFA accepts an input symbol α if and only if there is a path in the transition graph from the start state to some final state, such that the edge labels along this path spell out α. It is entirely legal to transition to several different states given the input α in some state N. In addition, epsilon (ϵ) transition is allowed in NFA. That means there may be a transition from a state to another with no input.

A DFA is a special case of an NFA, represented by a 5-tuple, (Q, Σ, Δ, q0, F), in which:

- No state with ϵ-transition.
- There is almost one transition α leaving state N for each state N and input α.

Every finite state automaton comes to mean a set of rules. There is another notation of representing these rules called regular expression (REGEX or REG). A REG is a description of a set of strings, i.e., a language. They can be used to detect occurrences of these strings. Syntax of REG can be seen in Table 2.1.

Character	**Sample REG**	**Description**
?	Kath?mandu	Matches the preceding character or subexpression zero or one time.
*	ba*	Matches the preceding character or subexpression zero or more times.
+	ba+	Matches the preceding character or subexpression one or more times.
\| | car\|bus | Matches either car or bus. |
( ) | cit(y\|ies) | Matches either city or cities. |

Table 2.1: A table of REG syntax

A finite state machine can be viewed as a structure almost like a directed graph. The states are just the nodes of the graph, and the state transition function defines edges. The start state of the machine corresponds to the initial node; the states that can be reached by an arrow from a given state are the neighbors of that state. But there are some important differences between directed graphs and automata. For instance, a directed graph could be cyclic or acyclic, and there is no obligation that there must be one or more finish nodes in a directed graph. However, an automaton has to have at least one final (accepted) state even though it is a cyclic automaton. So, an automaton is an rule-bound collection of states with connection among them, and it refers to a directed graph.

The visual-based approach is the most convenient way of comprehending graphs and network structure since graph visualization [16] brings off analyzing a set of data which has relations (transitions) between states even if they are large and complex networks.

2.2 Learning Theory

It is certain that any artefacts which are as complex as today's software systems have not been created up until now. As a result, not only creating, maintaining, but also teaching software is exactly a challenging task [15], [11]. Software visualization can be used to graphically represent various concepts in computer science education

at this point. Numerous computer science educators suggest an extensive belief that software visualization positively impacts learning. Many software visualization tools are developed to help to understand, test, or debug software both in education and industry.

Algorithm animation and program animation are two of the subfields of software visualization [18], [14]. Program animation provides users or learners to analyze programming constructs, e.g., instances or methods. Algorithm animation touches on the flow of program. It helps students learn the algorithms more effectively and consequently. The use of these types of visualization tools in education which is developed for algorithm animation has gained a lot of interest during recent years.

It is a common belief of computer science educators that algorithm animation can greatly benefit learners and instructors. With respect to educational effectiveness, it makes sense to look more closely at the experimental process. Visualization of algorithms has been successfully used to actively engage students in constructing their own input data, making predictions regarding future visualization states, programming the target algorithm, answering strategic questions about visualization, constructing their own visualizations, and receiving immediate feedback [12].

In computer science, teachers often use visualization and animation tools. For example, they use visualization when teaching data structures, sorting, graph theory, etc. It is aimed to facilitating understanding and learning an algorithm and the same for finite automata. Using visualization in finite automata teaching will provide hands-on experimentation of automata theory concepts, such as generating and visualizing process of a lexical analysis, creating a finite automaton from regular expression, converting a non-deterministic finite automaton to a deterministic one, or minimizing a deterministic finite automaton.

2.3 Animation in Learning Systems

Pictures are perceived by instinct of human while numerical data is comprehended by skills obtained from training at schools, in business life, etc. Therefore, lots of people are still not that good with numerical data. Visualization is the graphical presentation of information, with the goal of supporting learners in a better and more qualitative understanding of the information contents, and it is mostly easy to understand and analyze pictures. If there is a well-organized drawing, visualization or an animation, one is much easier to find trends and relations. Because visual presentation of information takes advantage of the vast and often underutilized, capacity of the human eye to detect information from pictures, diagrams, and illustrations.

Surveys of computer science educators suggest a widespread belief that visualization and algorithm animation positively impacts learning [21]. Algorithm animation has been successfully used to actively engage learners in constructing their own input data sets, making predictions regarding future visualization states, programming target algorithm, answering strategic questions about the visualization and constructing their own visualizations [22].

Visualizations and animations are related to multimedia. But there is a difference between visualization and multimedia research: interaction. A learner may have a chance to manipulate and affect the content of animation. According to Park and Hopkins [19], animations in learning systems have six educational advantages in comparison with textual representation which are:

- demonstrating sequential actions in a procedural task,
- simulating causal models of complex system behaviors,
- explicitly representing invisible system functions and behaviors,
- illustrating a task which is difficult to describe verbally,
- providing a visual analogy for an abstract and symbolic concept,
- obtaining attention focused on specific tasks or presentation displays.

At this point, the question is how effective such animation and visualization tools in the learning process are? Animations have to meet some necessities so that they may be put into service effectively. A concrete animation provides resources that help learners interpret the graphical representation by explaining the relationships. Novice learners can become quickly overwhelmed by too many details or windows, and they usually prefer to test an animation with predefined or basic input. Also providing the learner with multiple views (for example, The SHALEX System [13]) can facilitate a better understanding of the algorithm. As implemented in GaniFA NG, execution control mechanism plays a key role in the animation. Forward or backward flow controls ease the usage of the visualization tool. It is as important that animation should support learner-built visualizations. Allowing the learners to use their own input data engages them more actively in the animation process.

In spite of some trivial problems to be solved, the future of interactive, animated educational software supporting computer science education looks bright. Theoretical courses, such as operating systems, computer architecture, artificial intelligence, compilers, and networks, may come to life with more effective animation standards.

2.4 Visualization Background

In this thesis, the JUNG graph library [25] is used for modeling, analyzing, and visualization of FSAs. JUNG is a free and Java-based open source software library that is used for the analysis, manipulation, and the visualization of graphs and networks (an example can be seen in Figure 2.5). JUNG has a powerful structure to represent graphs so that it is pretty easy to visualize the data which is generated from a regular expression or by interactive design. It provides different layout algorithms or enables users to create custom layouts suited for their needs. Features that JUNG supports are:

- Representation of directed and undirected graphs, graphs with parallel edges, and relations among the graph nodes (states).
- Random graph generation, graph theory, exploratory data analysis.
- Interactive visualization.
- Default and custom layout algorithms.

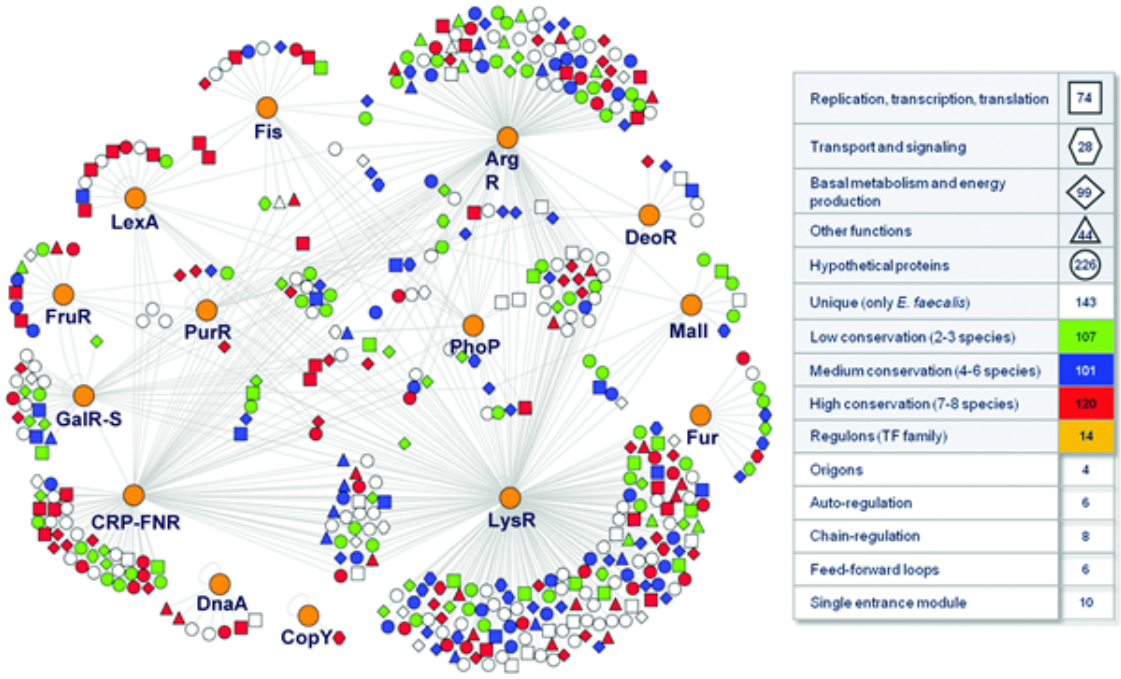


Figure 2.5: A sample graph view of JUNG Library

In general, visualization requires a layout, a component, and a renderer. JUNG provides *AbstractLayout*, *VisualizationViewer*, and *Renderer* classes for these. The default implementation fetches the location of vertices from the *Layout*, paints each one with the *Renderer* inside the *Swing* component. *VisualizationViewer* maintains the mouse behaviors and some other visualization details. In this thesis, these structural classes were extended and adapted, so that some special features could be achieved. Also graph, vertex, and edge operations were implemented in a way of extending the default object-oriented code architecture of JUNG.

JUNG currently provides many of the tools and elements that are most commonly required for writing software that manipulates, analyses, and visualizes network data sets [8]. Future releases are planned to add new analysis tools, support for new filtering methods, new visualization architecture, and creating standard XML-based representation.

JUNG also includes implementations of a number of algorithms from graph theory, data mining, and social network analysis, such as routines for clustering, decomposition, optimization, random graph generation, statistical analysis, and calculation of network distances, flows, and importance measures.

The JUNG architecture is designed to support a variety of representations of entities and their relations, such as directed and undirected graphs, multi-modal graphs, graphs with parallel edges, and hypergraphs. The directed graph feature is applied in this thesis.

3 Related Works

The study of finite state automata is pervasive in Automata Theory courses, and it is also a concept that is difficult for students to visualize with traditional materials with paper-and-pencil or on whiteboard. Thus, remarkable effort has gone into developing educational visualization software that provides more interactivity, and instant feedback in learning experience of automata theory in recent years. A few of them will be discussed in this chapter. Although a number of instructional visualization tools for experimenting with automata were developed, most of them suffer from one or more defects that make them less than ideal as learning tools, particularly for less advanced students.

jFAST [27], is a portable visualization application which is implemented purely in Java, using Swing for all user interface components allows user design, modify, and visualize FSAs. jFAST provides a designer, which is shown in Figure 3.1, for drawing automata, such as OpenOffice Draw. It uses feature handler and component interface in order to accomplish visualization issues. It has a design error checking feature that analyses the current automaton and provides an error message about the mistakes found with suggestions. jFAST has an interesting feature called sub-automata integration. An automaton can be created in a number of sub-automata, each of which is a distinct automaton. These sub-automata can then be merged to form more complex FSAs. A good feature of jFAST is the ability of sending created automata to students via email. However, its layout architecture is not enough to represent an automaton actually. Cross paths or parallel edges are big flaw for jFAST.

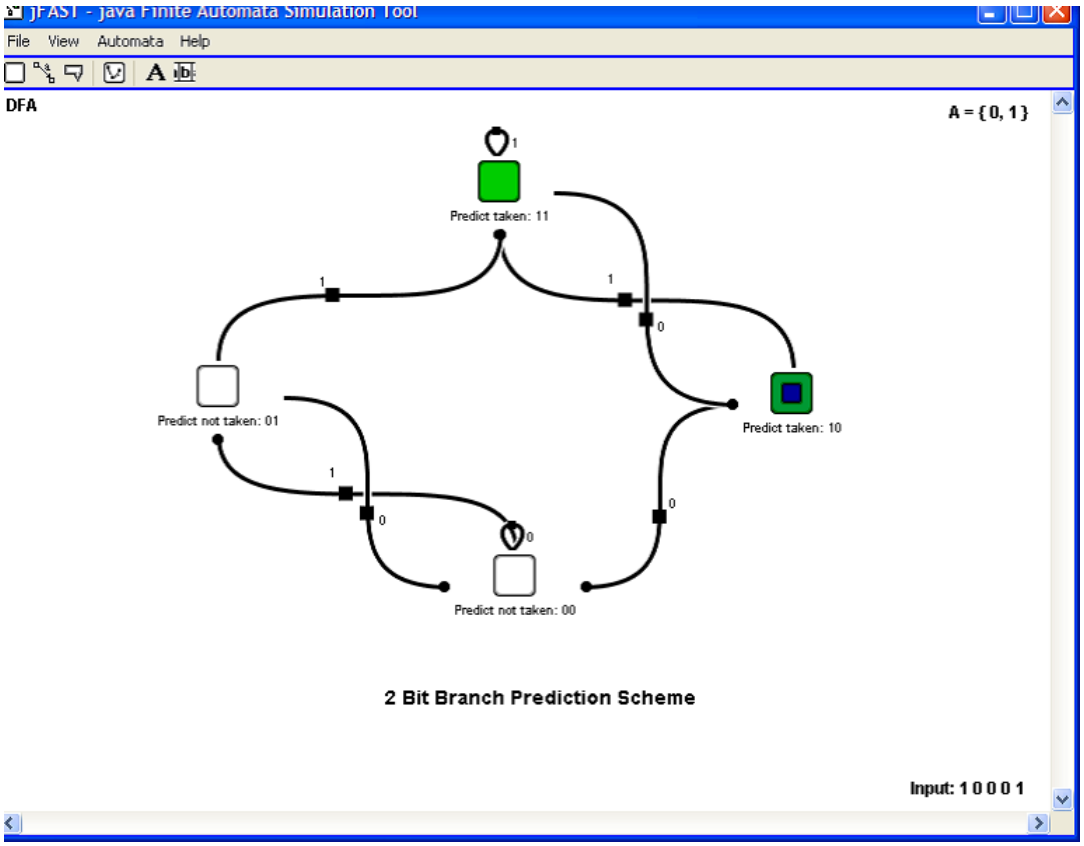


Figure 3.1: jFAST (Java Finite Automata Simulator)

ASSIST [9], is an instructional tool and a finite state automata simulator for formal language learners. However, it is difficult to setup and use. To use the simulator, an understanding of the formal definition of each category of language which is put upon is necessary.

JFLAB [23] can also be considered that it has a bit complicated screen design for ordinary students with less knowledge about finite state automata. Also JFLAB do not focus its work on the visualization and layout of the finite automata, thus, the available layout algorithms are very basic and simple.

GUItar is a graph visualization tool which provides multiple pages so that user can visualize multiple automata. It is possible to collapse multiple arcs between two states [1]. There is also a node style manager for editing a style that could be used to represent a state which is start, final, etc. GUItar has an impressive user interface and a handy tool bar menu for basic operations. Interactive graph drawing, changing style of nodes and edges, editing layout algorithm, import and export are possible in GUItar (Figure 3.2). Therefore, automatic automaton drawing is being planned in the near future.

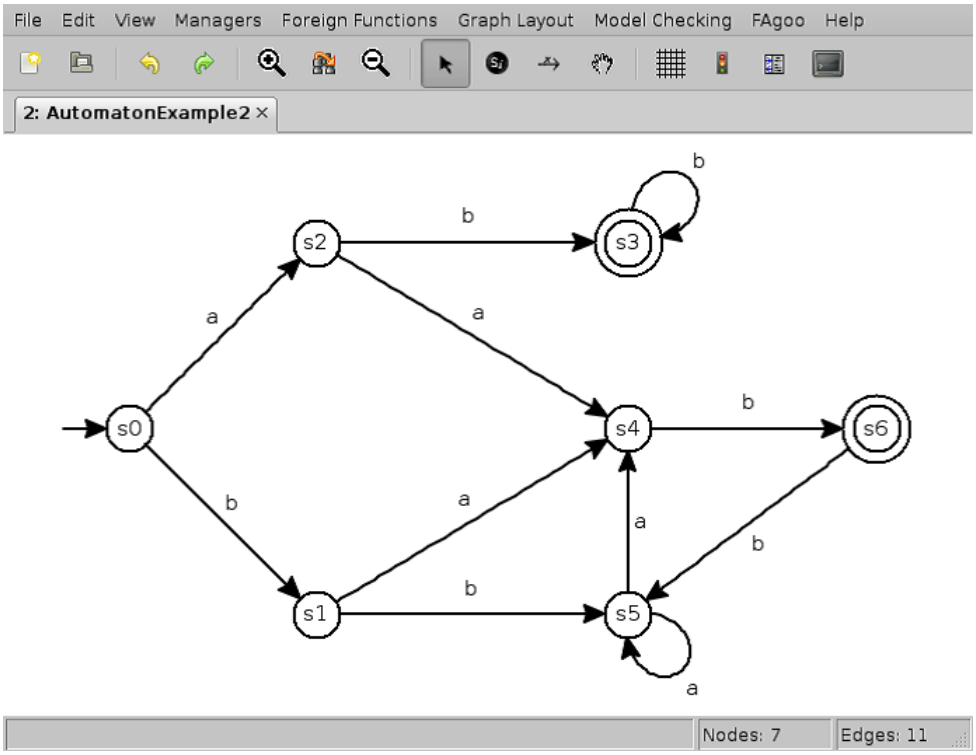


Figure 3.2: GUItar

PetC [2], a master thesis tool is not sufficiently clear for using and does not have dynamic simulation capabilities in detail.

Visual Automata Simulator [3], is designed for advanced students by assuming that they have already heard about the basic concepts. Especially Turing Machine module can be seen as distractive.

The old GaniFA [6] system was developed at the end of 1990s. It was compatible with Netscape browser and web standards of this time. In principle, GaniFA NG and the old GaniFA are replicas to each other. The old GaniFA has all basic capabilities GaniFA NG performs. These are generating NFA from REG, ϵ-transition

removing, NFA to DFA converting, NFA minimizing, and visualization of automaton computation overall. It was included in a web page as applet and based on foresighted layout [7] [4]. A screenshot from old GaniFA can be seen in Figure 3.3.

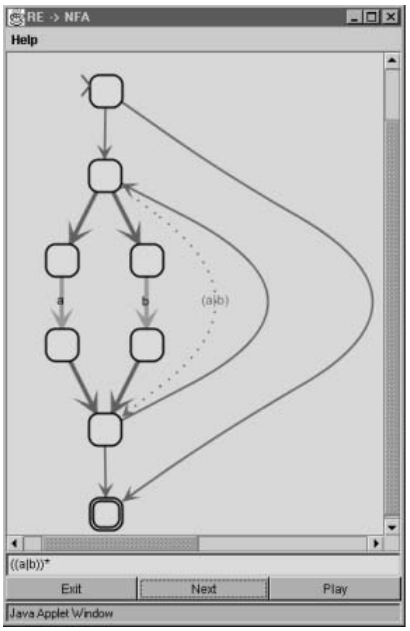


Figure 3.3: Old version of GaniFA

Despite the fact that all the tools above are visualization tools, the GaniFA NG tool which is presented in this thesis has a remarkable advantage that brings it forward than the others. The GaniFA NG directs the user to follow the use case. This facilitates the pursuit and guides the user during the computation of processes.

The GaniFA NG tool provides a robust and smooth interface and a plain scenario. It incorporates the facilities of editing and visualizing automata. The graphical interface basic frame is composed by a menu bar and a designer panel. Users have a chance to work with multiple automata at the same time with multiple tabs, each one containing a different type of automaton. The designer panel module provides users to create or manipulate automata with mouse events. There are a set of mouse events for creating nodes, creating transactions (edges), setting as final or start node, and removing any object added before. GaniFA NG distinctly directs the users during the usage with its menu bars. For instance, a user can only create a NFA. If he/she wants to create a DFA and save it in order to use in the future, he/she has to start to create NFA. Before the user can create a new NFA with no state, the rest of the tabs and menu bars are disabled, so the user should know that it is compulsory to launch the NFA creation. Also it is possible to visualize multiple different type automata at the same time with any loss of data. Every mouse event or other manipulation steps are realized with a thread-based precedence structure.

As a consequence, GUItar holds upper hand against the rest of the applications mentioned above. But, GaniFA NG has a more sophisticated menu diversity.

4 Analysis and Design

In this chapter, the general design and graphical user interface of GaniFA NG will be introduced.

4.1 Conceptual Approach

The construction and analysis of algorithms and computations is one of the fundamental areas in computer science. Visualizations are convenient for learning algorithms and computations due to the abstraction. Educational software systems are designed for students to learn theoretical computer science, e.g., automata theory and formal semantics and aims to provide highly gifted environments during the computation of complex processes. But they diverge from each other in methodology and efficiency. According to Diehl and Kerren, the old GaniFA system realizes the "Second-Order Generative Approach" [5], since it allows users to compute and generate their automata with an interactive visualization.

In this thesis, an enhanced version of the old GaniFA tool which is named as GaniFA New Generation (shortly GaniFA NG) is introduced. GaniFA NG has been developed for experimenting with concepts in formal languages and automata theory just as the old GaniFA tool.

The visualization gives an alive and descriptive feedback during the evaluation and computation of algorithms. The goal of our tool architecture is to provide an easy-to-use platform for learning about FSAs through interactive design and graphical simulation.

4.2 Graphical User Interface

GaniFA NG has been designed with a graphical user interface like the old GaniFA. The main visual innovation in GaniFA NG is the tabbed view. The general view of GaniFA NG looks like a browser with six stable tabs. Four tabs are used for FSA simulation (NFA, DFA, Min-NFA, and Min-DFA).

- NFA Tab: This tab is actually one of the first access points of the tool with the DFA tab. The user should use a REG or load a saved NFA if he/she wants to run GaniFA NG.

- DFA Tab: GaniFA NG does not provide DFA creation, so the user has to create an equivalent NFA first. This ensures absolute user achievement. Alternatively, the user is able to load a saved DFA and run GaniFA NG.

- Min-NFA Tab: The Minimized NFA tab becomes active only when the user desires to remove ϵ-transitions of a NFA.

- Min-DFA Tab: This tab is enabled when the other conversion scenario is handled.

- Help Tab: The user may need some directives and guidance for the tool. Explanation and instructions contained are in this tab.

- About Tab: The About tab gives an outline about the concepts of GaniFA NG.

In a first version of GaniFA NG, the main page was divided into four sections instead of tab views, but the users would have difficulties to study FSA because of lack of space. Besides this, there is no need to show all windows at the same time. In connection to this, GaniFA NG directs the user to the relevant tab according to his/her reaction during the visualization. The top and the bottom menus of each tab present a standard design template with the general visualization buttons.

In each FSA visualization tabs (NFA, DFA, Min-NFA, and Min-DFA) all nodes can be dragged and the FSA can be redesigned from the point of appearance. A visualization may be little hard to understand for the user, and he/she is able to adjust FSA in order to smooth it.

4.3 Interaction Design

GaniFA NG does not use static REG for a NFA creation or static input text for the FSA visualization. The users are able to run the tool with his/her own REG or input text easily. They (especially the learners) see how an algorithm works; as well why the algorithm works. They may prefer to create empty NFAs. GaniFA NG acts as a successful advisor in this case. For instance, the users are obligated to naming and entering a right input, while creating a node or an edge.

5 Implementation

In this chapter, implementation aspects and issues of GaniFA NG will be described. The chapter also includes a brief description of major components, platform, and environment in which the codes are written and tools that are used for the implementation.

5.1 Implementation Details

In this chapter, implementation details will be described. Instead of explaining programming codes line by line, an architectural method will be tracked.

5.1.1 Language, Tools, and Libraries

Programming language, external libraries, development environment (IDE), and operating system, etc. of GaniFA NG are as listed in Table 5.1:

Programming Language	JAVA
External Library	JUNG
Database	No database used in this thesis
Operation System	Windows 7 x64
IDE	Eclipse Standard SDK version: Kepler Release

Table 5.1: A table of language, tools, and libraries

5.1.2 Software Development Architecture

Since the purpose of GaniFA NG is retrieving FSA data from users in various ways and displaying a proper representation in return, the system should realize user changes impeccably. The Model-View-Controller (MVC) pattern which is shown as schematically in Figure 5.1 is applied as a basis in order to overcome this issue in this thesis.

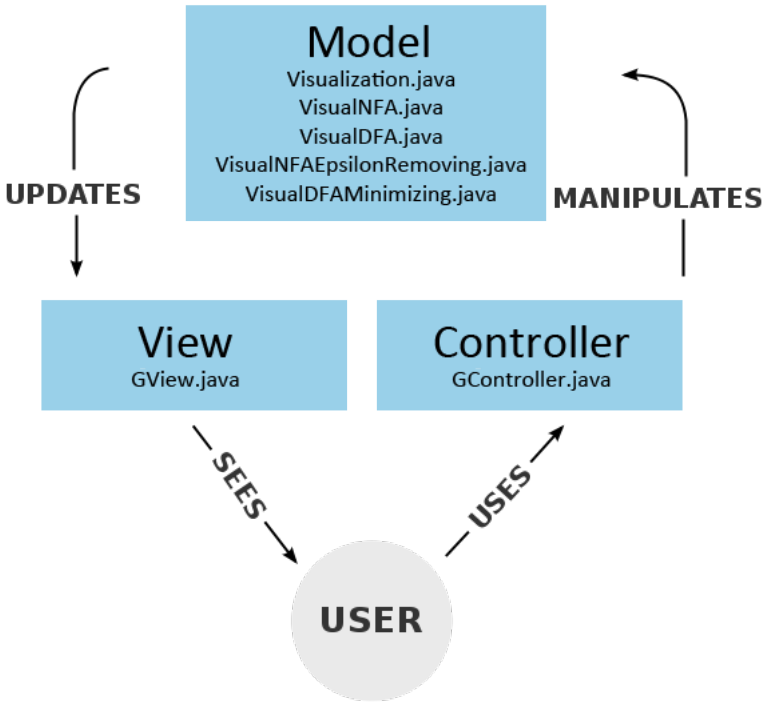


Figure 5.1: Model-View-Controller implementation structure of GaniFA NG

The project starts with *GMain* in ganifa.main package. This class calls *GFrame* which extends *JFrame*. MVC structure is based on *GController* and *GView* in ganifa.main, and all classes inherited from *Visualization* class in ganifa.visualization package. *GView* observes the changes from *Visualization* class and applies immediately.

The class structure can be seen in Figure 5.2. Each class inherited from *Visualization* consists of a *FiniteAutomaton*, a *GanifaLayout*, a *Viewer*, and a *GraphMouse* object. These object types will be described at the occasion arises.

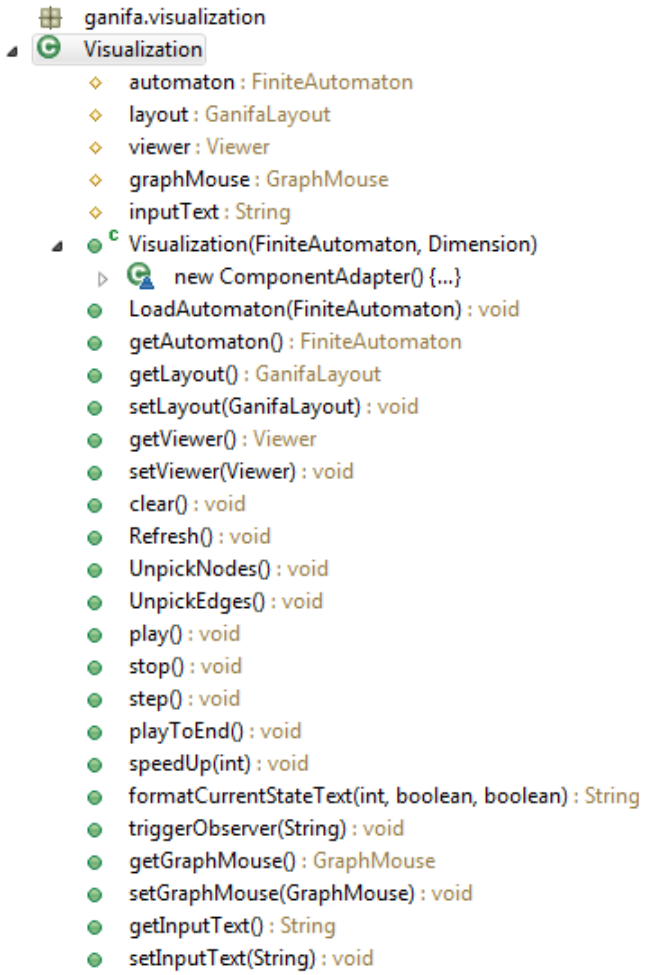


Figure 5.2: Visualization.java class structure

All classes with their packages can be seen in Figure 5.3 with a package explorer screenshot. ganifa.automaton includes *AbstractFiniteAutomaton* and *FiniteAutomaton*. *FiniteAutomaton* extends from *AbstractFiniteAutomaton* and is used as a generic class of the *NFA* and the *DFA*. The packages of ganifa.automaton.nfa and ganifa.automaton.dfa consist of their related automaton type classes *NFA*, and *DFA*. *NFAMinimizer* and *DFAMinimizer* are used for automata conversion. Also *NfaToDfaConverter* takes the current NFA as a parameter and practices the conversion process.

ganifa.automaton.regex consists of *Alphabet*, *Parser*, and *RegularExpression*. *RegularExpression* has a global object in type of Expression class. A State-Strategy design pattern is used here. *Expression* is located in ganifa.automaton.expression package, and it is an abstract class with Alphabet. The rest of *Expression* originated classes in this package is used with a Decorator design pattern structure. As it can be seen in Figure 5.4, *Expression* has an abstract method called BuildNFA(). This method is overridden in extended classes according to their missions in a REG.

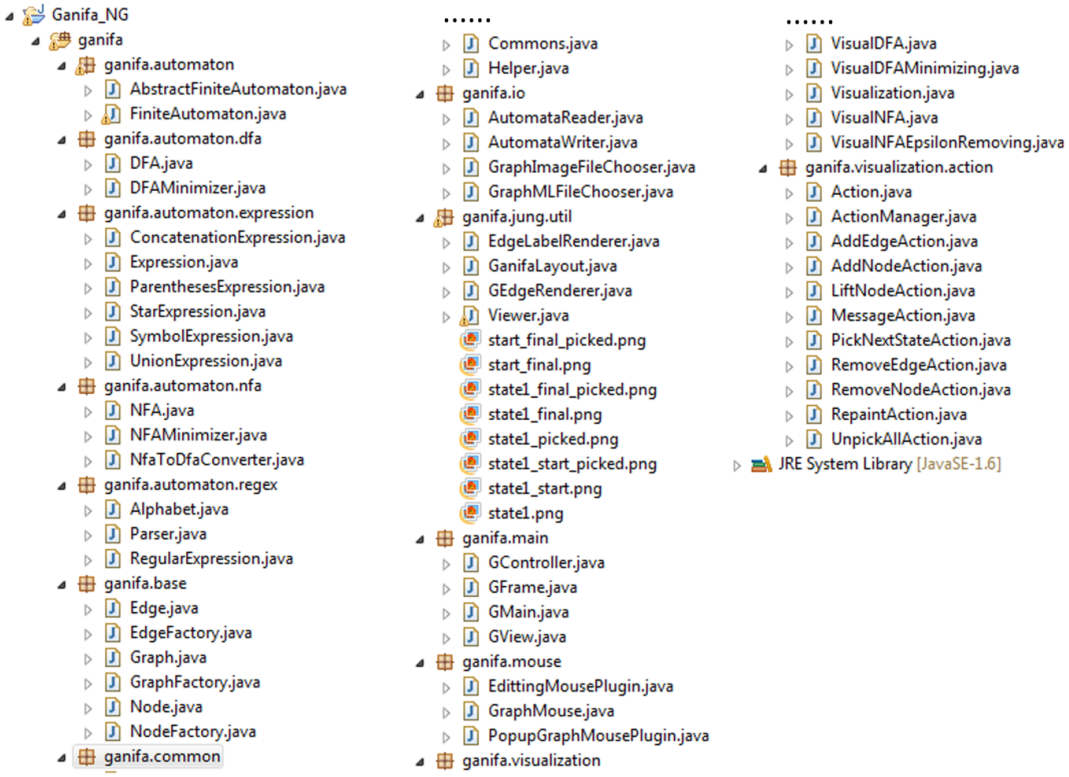


Figure 5.3: Package explorer screenshot of project

```
1  package ganifa.automaton.expression;
2
3  import ganifa.automaton.nfa.NFA;
5
6  public abstract class Expression {
7
8      protected Alphabet alphabet;
9
10     public Expression() {
11     }
12
13     public Expression(Alphabet pAlphabet) {
14         this.alphabet = pAlphabet;
15     }
16
17     public abstract NFA BuildNFA();
18
19     public Alphabet getAlphabet() {
20         return alphabet;
21     }
22
23     public void setAlphabet(Alphabet alphabet) {
24         this.alphabet = alphabet;
25     }
26 }
```

Figure 5.4: Expression class

ganifa.common package includes Commons class for static variables used in whole project and *Helper* that is applied for a few FSA tasks. ganifa.io package is formed by graph (FSA) reader and writer classes which are *AutomatonReader* and *AutomatonWriter*. Both classes use a GraphML file format as shown in Figure 5.5.

```
<?xml version="1.0" encoding="UTF-8"?>

<graphml xmlns="http://graphml.graphdrawing.org/xmlns/graphml"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns/graphml">

    <key id="type" for="graph">
        <desc>Automaton Type</desc>
    </key>
    <key id="position" for="node"/>
    <key id="isStart" for="node"/>
    <key id="name" for="node"/>
    <key id="isFinal" for="node"/>
    <key id="transaction" for="edge"/>
    <graph edgedefault="directed">
    <data key="type">NFA</data>
        <node id="n0">
            <data key="position">0.000000;0.000000</data>
            <data key="isStart">True</data>
            <data key="name">n0</data>
            <data key="isFinal">False</data>
        </node>
        <node id="n1">
            <data key="position">0.000000;0.000000</data>
            <data key="isStart">False</data>
            <data key="name">n1</data>
            <data key="isFinal">True</data>
        </node>
        <edge source="n0" target="n1">
            <data key="transaction">a</data>
        </edge>
    </graph>
</graphml>
```

Figure 5.5: An example of GraphML file format.

ganifa.base serves as a core package while automaton data structure is being constructed. Because pure automaton class called *FiniteAutomaton* uses an object of type *Graph*. Also *Edge* (transition) and *Node* (state) classes are found here. ganifa.util is base (utilization) package of visualization issues, as it is understood from its name. Classes of this package are customized classes of JUNG library, and they work for drawing the shapes of visualization elements like nodes or transitions. Additionally, these kinds of things are happened here: how much curved an edge will be, or where a label of node will be located. ganifa.mouse package manages mouse actions during automaton manipulation.

ganifa.visualization.action package has a class called *ActionManager* that directs the visualization process. Each visualization event is performed by an Action extended class (*AddEdgeAction*, *AddNodeAction*, *PickNextstateAction*, etc.). All action classes are put in an *ArrayList<Action>* called activities declared in *ActionManager*. When the user triggers the visualization all the actions take the stage one

by one in a Thread with a time gap. There are some other fields of design pattern applications can be seen in Figure 5.6.

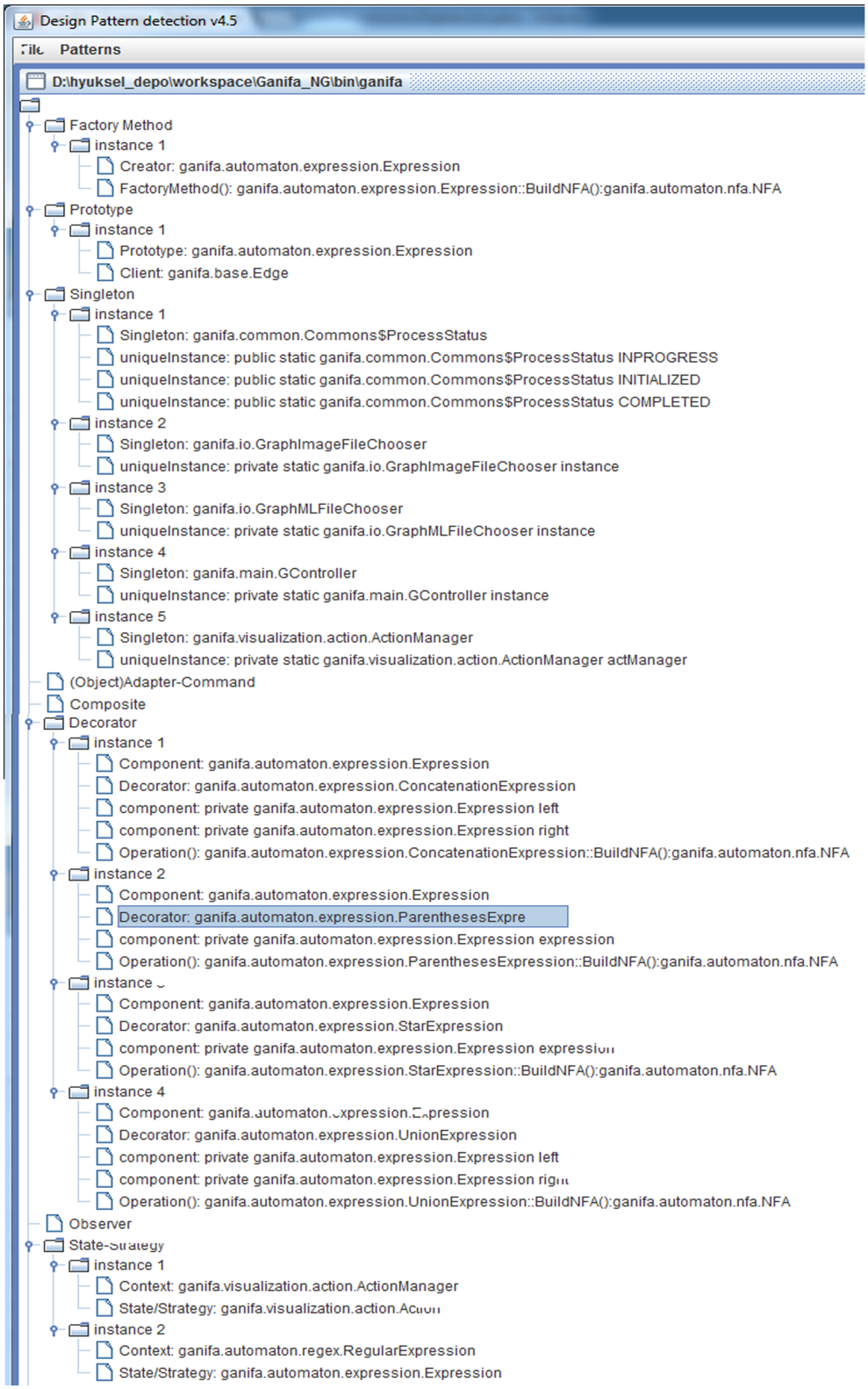


Figure 5.6: Design patterns applied in GaniFA NG

6 GaniFA NG - Tool

The old GaniFA was primarily developed at the end of 1990s with Java. GaniFA NG is also written in Java and utilizes the object-oriented design and principles of Java to derive a conceptual FSAs from the design, which is then used for simulating and testing the FSAs on different inputs as specified by the user. The project was envisioned as web-based software that brings learners experimenting with automata theory and finite state automata firstly. It included creating, manipulating, and visualizing all types of finite state automata. This new version not only serves the purpose but also provides new capabilities like drag-drop, multiple page visualization, etc. It allows to construct automata and run traces on input strings. The main window of GaniFA NG can be seen in Figure 6.1.

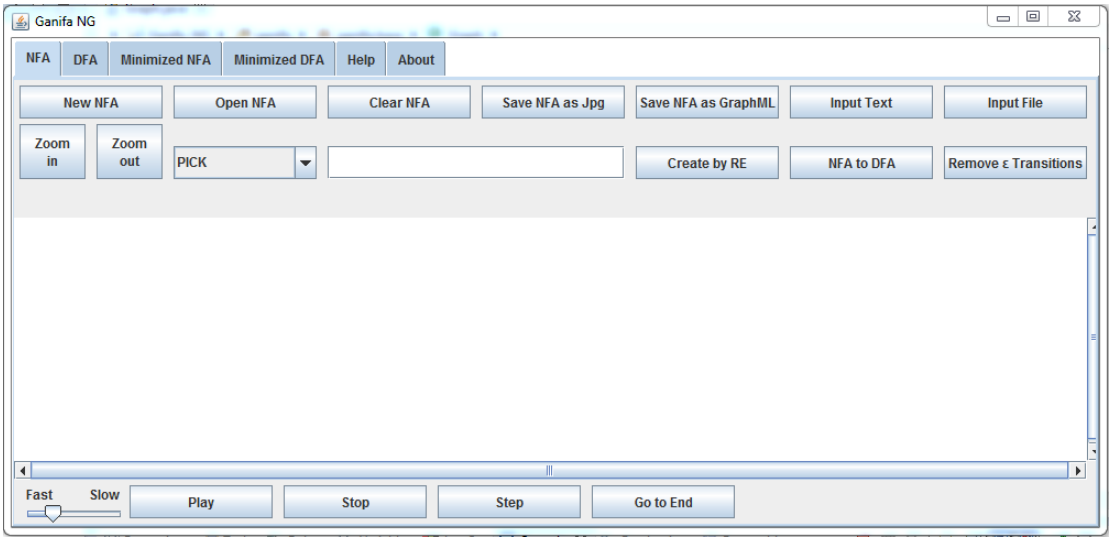


Figure 6.1: GaniFA NG Main Page. All tabs and menu buttons are actually not enabled. Users are directed to the right menus by enabling or disabling them during usage.

6.1 GaniFA Capabilities

GaniFA NG is able to achieve the following phases:

- Generating NFAs from REGs
- Removing ϵ-transitions from NFAs
- Converting NFAs to DFAs
- Minimizing DFAs
- Visualizing of computing FSAs from any input defined by user or the system

In this chapter, these capabilities will be described in detail and step by step.

6.1.1 NFA Capabilities

When the user starts GaniFA NG, the main page is shown up with four tabs. One of the enabled tabs is the NFA tab, and the user is provided NFA operations in this (first) tab. There are two options to start NFA operations: a new NFA can be created, or a saved NFA can be loaded from a directory. GaniFA NG uses GraphML for saving automata and loading them later if needed. GraphML [24] is a file format which is comprehensive, easy to use, and directed graphs are supported.

6.1.1.1 NFA Operations

- Create new NFA: The first step of automata visualization is NFA creation. The "New NFA" button both creates a new non-deterministic finite automaton with no states, and enables the top menu and sub menu items. All automata in GaniFA NG have an alphabet that consists of lower and upper case letters and digits. All other symbols are assumed invalid characters by the system.

- Load pre-saved NFA: If the user desires to open an NFA saved before, he/she clicks the "Open NFA" button. An open file dialog box appears and asks the directory of pre-saved NFA file with an extension of ".graphml". It is based on common XML structure. What information kept in a GraphML file is:

  - Automaton type (e.g., NFA or DFA)
  - Automaton edge type (e.g., directed, undirected)
  - States (or nodes)
    - Name
    - Position (e.g., 1.000000;4.000000)
    - Is start Node (e.g., true or false)
    - Is Final Node (e.g., true or false)
  - Edges (or transitions)
    - Transition input (one character string)

  When a user clicks OK, the NFA is loaded into the NFA panel, and the old automaton is removed if it exists.

- Clear NFA: The "Clear NFA" button removes all nodes and edges from the current automaton.

- Save as Image: Another automaton saving option is saving as image. The user can save the current automaton as an image by clicking "Save NFA as Jpg". This menu item opens a save file dialog, and the user determines the file name via this dialog. If there are no nodes and edges, a warning message appears, and the user cannot complete the saving process.

- Save as XML: The user has an option to save the NFA with an extension of ".graphml", so that he/she is able to use it in the future.

- Visualize Input Text: There are two ways of testing the input data in GaniFA NG (input text and input file). The user displays and visually simulates his/her own input text via the "Input Text" button, and analyzes the current NFA. He/she can also follow the steps of the visualization algorithm, so that he/she can see if the current transition is accepted by input text field located under the zooming buttons. A message box is shown with a corresponding message at the end of the process. The begin and end of the visualization process are presented in Figure 6.2 and Figure 6.3.

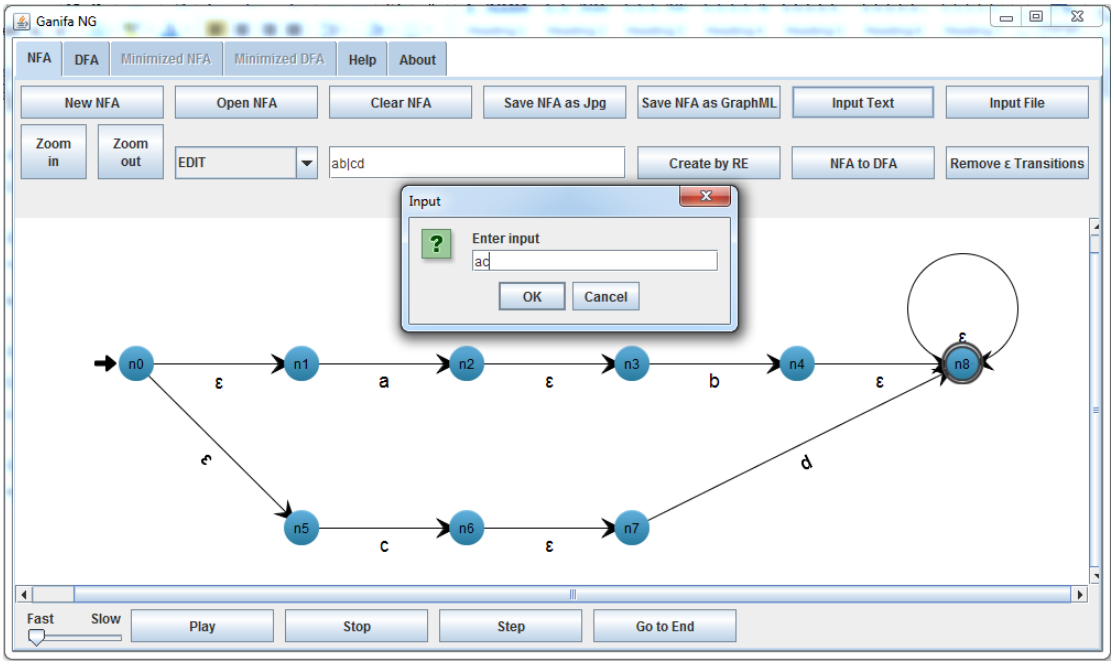


Figure 6.2: Entering an input string for automata visualization

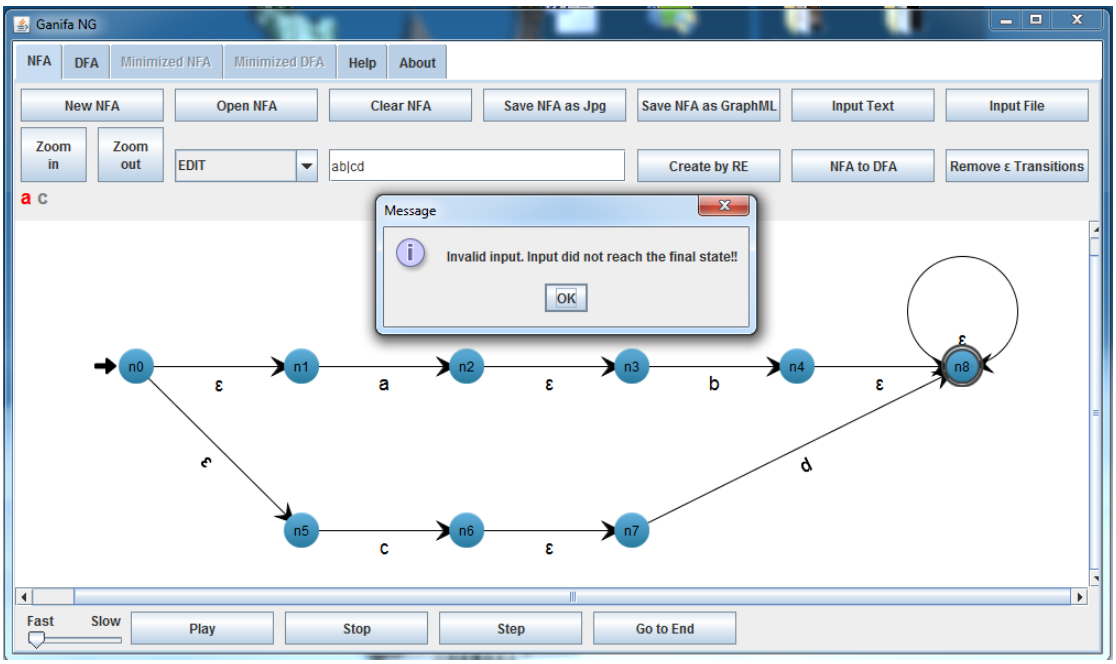


Figure 6.3: The visualization module simulates the input string on the left hand side of the tool bar.

- Visualize Input File: The user can visualize input available in a text file by clicking the "Input File" button. After determining the corresponding text input file, visualization process starts.

- Zoom in: "Zoom in" button available in the sub menu is used for zooming in the created NFA.

- Zoom out: "Zoom out" button available in the sub menu is used for zooming out the created NFA.

- Change NFA mode: There are three options of manipulating a NFA. There are:

  – Pick: Users can pick the created NFA, change the positions of node.

  – Edit: Users can add, remove, and edit nodes and edges.

  – Transform: This mode enables to move NFA completely.

- Create NFA by RE: The user enters a valid RE into the text box in the sub menu and clicks the "Create by RE" button. The RE is put in process by a parser. The RE should consist of symbols remarked by NFA alphabet. The parser divides the RE into smaller expressions according to the special symbols it detects. For instance, if a star symbol ("*") occurs, the parser reduces this part of expression and analyses separately. There are five types of expressions. These are concatenation, parentheses, star, symbol, and union expressions. If it encounters "*", this means that there is a star expression there. If a "|" comes up, then a union expression occurs. "(" and ")" are used for indicating parentheses expressions. Two or more symbol occurrences in sequence mean a

concatenation expression is parsed. When a unique symbol is detected during the parse process, two nodes and an edge is created in the NFA. Standard RE syntax rules are prevalent.

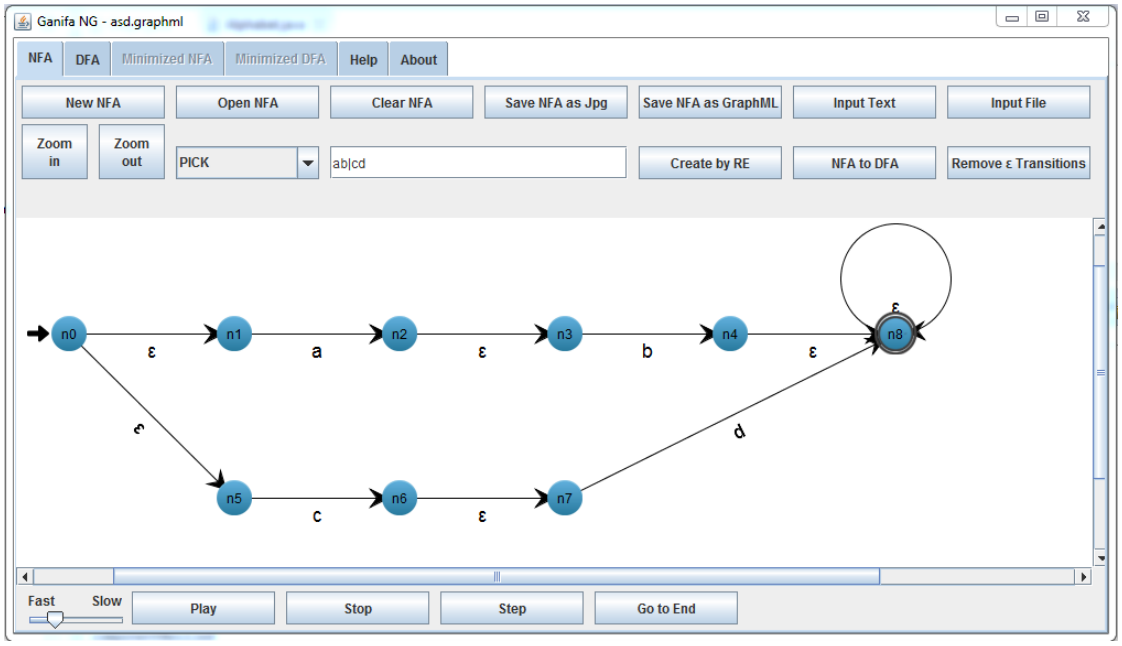


Figure 6.4: An example of NFA generation from a REG

- RE Samples: Some valid and invalid regular expressions can be seen in Table 6.1.

REG	Valid
abc	Yes
ba12	Yes
a*	Yes
A*	Yes
a**	No
(a)*	Yes
(a*)	Yes
a|b	Yes
ab|	No
*a	No
(a)|(b)	Yes
(ab*)|(c(d)*)	Yes

Table 6.1: A table of REG examples

- Convert NFA to DFA: The user can create an equivalent DFA by clicking the "NFA to DFA" button. The DFA panel becomes current pane, and the DFA starts to being visualized as shown in Figure 6.5.

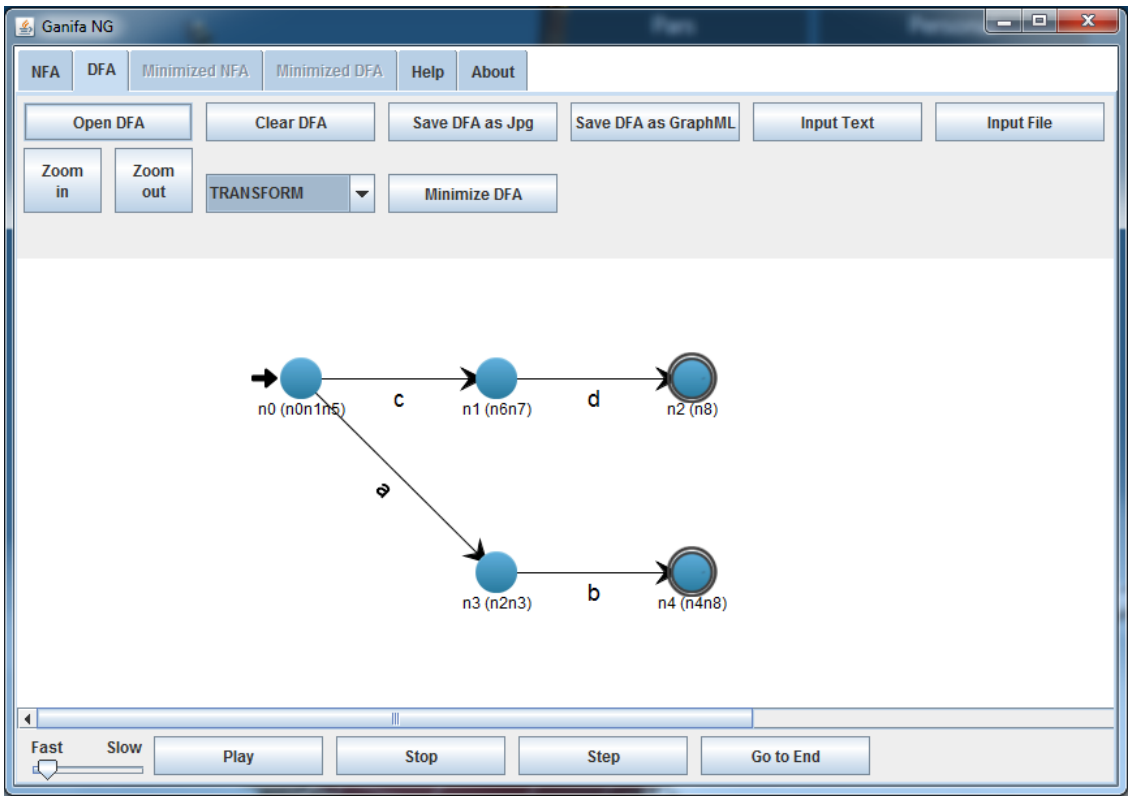


Figure 6.5: NFA to DFA conversion of regular expression ab|cd

- Remove ϵ-Transitions from NFA: The user can eliminate empty transitions by clicking the "Remove ϵ-Transitions" button and activate the minimized NFA panel. An equivalent Min-NFA will be created with the same REG.

- Set NFA Visualization Speed: The user can speed up or slow down every visualization takes place in NFA tab by using the slider in the left bottom corner of the page.

- Play NFA Visualization: If any visualization stops, the user can unpause it by clicking the "Play" button.

- Stop NFA Visualization: The user can suspend the visualization by the "Stop" button.

- Step NFA Visualization: It is a necessity to be able to visualize any automaton step by step so that user can perceive each simple action more comfortably. The "Step" button makes it possible.

- Go to End of NFA Visualization: It is possible to reach end of the visualization immediately by using the "Go to End" button.

- Speed Up and Slow down Visualization: A slider serves for adjusting the speed of the NFA simulation.

6.1.2 DFA Capabilities

DFA operations start in two different ways which are conversion of NFAs like in Figure 6.5 and loaded pre-saved DFA files as seen in Figure 6.6.

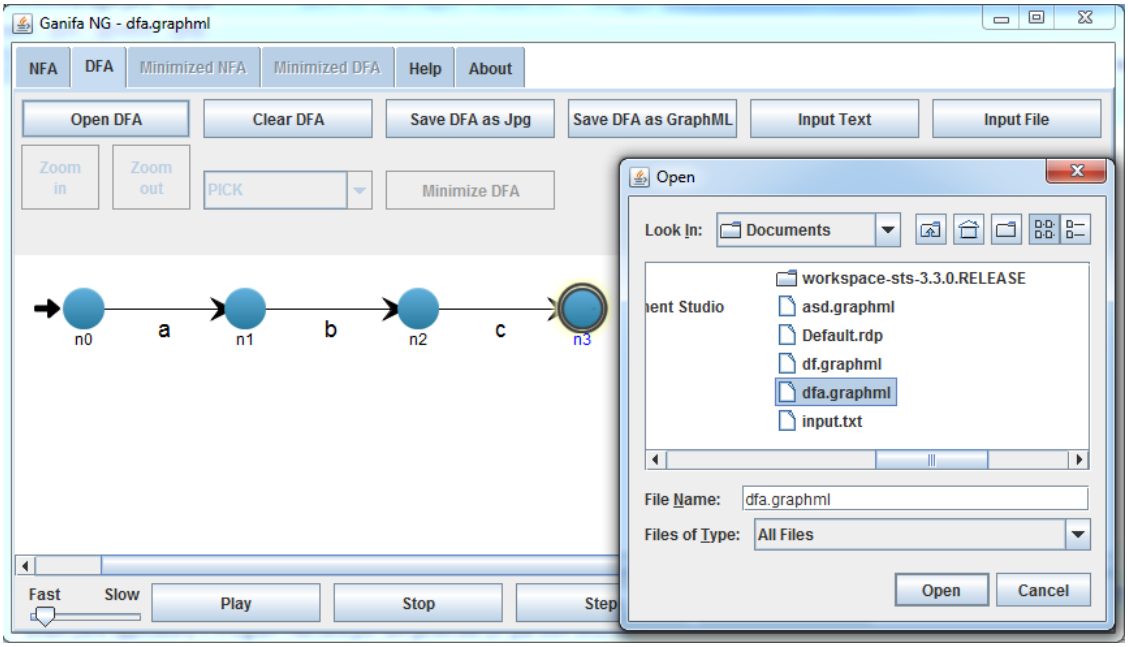


Figure 6.6: Pre-saved DFA and dialogue box of “Open DFA” button

6.1.2.1 DFA Operations

- Load pre-saved DFA: If the user desires to open a DFA saved before, he/she clicks the “Open DFA” button. A dialog box appears and asks the directory of pre-saved DFA file (graphml file).
- Clear DFA: After creating or loading a DFA, the “Clear DFA” button becomes enabled, and the user can remove all nodes and edges by using this menu item.
- Save as Image: The user can save his/her DFA as an image by clicking the “Save DFA as Jpg” button. This menu item opens a save file dialog, and the user determines the file name via this dialog.
- Save as XML: The user has an option to save his/her DFA as a GraphML file so that he/she is able to use it in the future.
- Visualize input text: The "Input Text” top menu item gives the user to visualize an input text by using his/her DFA. The user is given a message with the corresponding message at the end of the visualization.
- Visualize input File: The user can visualize the input available in a text file by clicking the “Input File” button. After determining the corresponding text input file, the visualization process starts.
- Zoom in: The Zoom in” button available in the sub menu is used for zooming in the created DFA.

- Zoom out: The “Zoom out” button available in the sub menu is used for zooming out the created DFA.

- Change DFA mode: There are three options of manipulating a DFA. There are;

  - Pick: The user can pick the created DFA, change the positions of a node.

  - Edit: The user can add, remove, and edit nodes and edges.

  - Transform: This mode enables to move the DFA completely.

- Create Minimized DFA: The user can create the corresponding minimized DFA by clicking the “Minimize DFA” button and activate the minimized DFA panel.

- Set DFA Visualization Speed: The user can speed up or slow down every visualization takes place in the DFA tab by using the slider at the left bottom corner of the page.

- Play DFA Visualization: If any visualization stops, the user can restart it by clicking the “Play” button.

- Stop DFA Visualization: The user can suspend the visualization by the “Stop” button.

- Step DFA Visualization: It is a necessity to be able to visualize any automaton step by step so that the user can see each action more comfortably. The “Step” button makes it possible.

- Go to End of DFA Visualization: It is possible to reach end of the visualization immediately by using the “Go to End” button.

- Speed Up and Slow down Visualization: A slider serves for adjusting the speed of the DFA simulation.

6.1.3 Minimized NFA Capabilities

The Min-NFA panel is activated when ϵ-transition removing operation starts in the NFA panel. The generated Min-NFA can be seen in Figure 21.

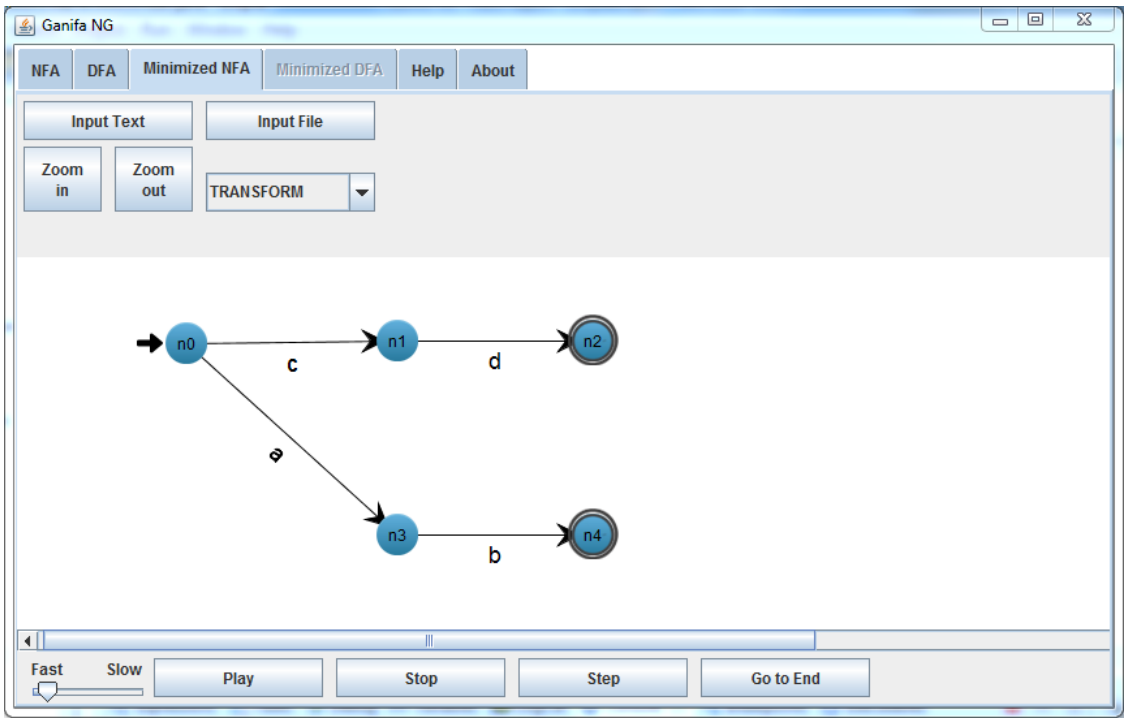


Figure 6.7: A minimized NFA generated from a NFA

6.1.3.1 Minimized-NFA Operations

- Visualize input text: The "Input Text" top menu item gives the user to visualize an input text by using his/her Min-NFA. The user is given a message with the corresponding message at the end of the visualization.
- Visualize input File: The user can visualize the input available in a text file by clicking the "Input File" button. After determining the corresponding text input file, the visualization process starts.
- Zoom in: The "Zoom in" button available in the sub menu is used for zooming in the created Min-NFA.
- Zoom out: The "Zoom out" button available in the sub menu is used for zooming out the created Min-NFA.
- Change Min-NFA mode: There are three options of manipulating a Min-NFA. There are;
  - Pick: The user can pick the created Min-NFA, change the positions of a node.
  - Edit: The user can add, remove, and edit nodes and edges.
  - Transform: This mode enables to move the Min-NFA completely.
- Set Min-NFA Visualization Speed: The user can speed up or slow down every visualization takes place in the Min-NFA tab by using the slider at the left bottom corner of the page.

- Play Min-NFA Visualization: If any visualization stops, the user can restart it by clicking the "Play" button.
- Stop Min-NFA Visualization: The user can suspend the visualization by the "Stop" button.
- Step Min-NFA Visualization: It is a necessity to be able to visualize any automaton step by step so that user can see each action more comfortably. The "Step" button makes it possible.
- Go to End of Min-NFA Visualization: It is possible to reach end of the visualization immediately by using the "Go to End" button.
- Speed Up and Slow down Visualization: A slider serves for adjusting the speed of the Min-NFA simulation.

6.1.4 Minimized DFA Capabilities

When the user minimizes a DFA generated in the DFA panel, the minimized DFA panel is activated, and a corresponding Min-DFA is created like in Figure 6.8.

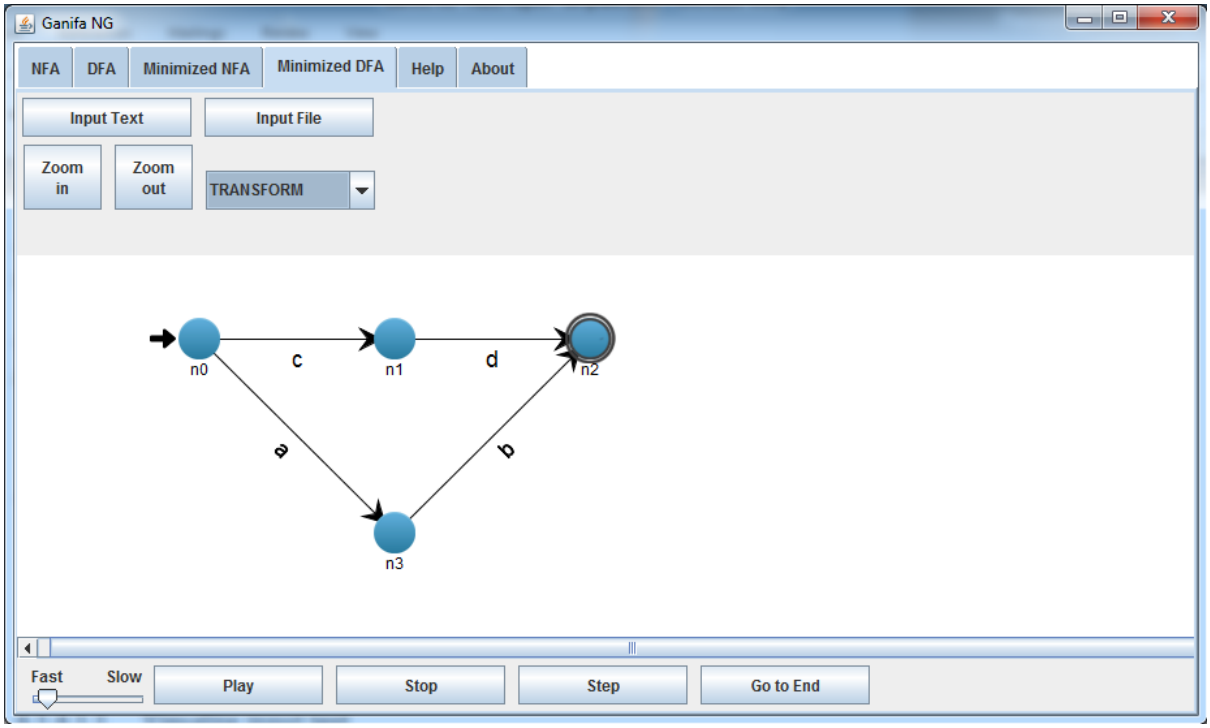


Figure 6.8: A minimized DFA generated from a DFA

6.1.4.1 Minimized-DFA Operations

- Visualize input text: The "Input Text" top menu item gives the user to visualize an input text by using his/her Min-DFA. The user is given a message with the corresponding message at the end of the visualization.
- Visualize input File: The user can visualize the input available in a text file by clicking the "Input File" button. After determining the corresponding text input file, the visualization process starts.

- Zoom in: The “Zoom in” button available in the sub menu is used for zooming in the created Min-DFA.

- Zoom out: The “Zoom out” button available in the sub menu is used for zooming out the created Min-DFA.

- Change Min-DFA mode: There are three options of manipulating a Min-DFA. There are;

  - Pick: The user can pick the created Min-DFA, change the positions of a node.

  - Edit: The user can add, remove, and edit nodes and edges.

  - Transform: This mode enables to move the Min-DFA completely.

- Set Min-DFA Visualization Speed: The user can speed up or slow down every visualization takes place in the Min-DFA tab by using the slider at the left bottom corner of the page.

- Play Min- DFA Visualization: If any visualization stops, the user can restart it by clicking the “Play” button.

- Stop Min- DFA Visualization: The user can suspend the visualization by the “Stop” button.

- Step Min- DFA Visualization: It is a necessity to be able to visualize any automaton step by step so that the user can see each action more comfortably. The “Step” button makes it possible.

- Go to End of Min-DFA Visualization: It is possible to reach end of the visualization immediately by using the “Go to End” button.

- Speed Up and Slow down Visualization: A slider serves for adjusting the speed of the Min-DFA simulation.

7 Conclusions

In order to extend the ideas and application, in this chapter, a conclusion of the work with the final outcome will be discussed together with the future work. This will help us to get a clear and detailed explanation of what has been done so far and what can be developed further.

7.1 Discussion

In Chapter 2, the thesis introduces the basic knowledge of automata theory and finite state automata. It also describes related concepts using basic knowledge of them, for instance, learning theory and algorithm animation. The aim of this thesis is using visualization concepts and animation capabilities to upgrade old GaniFA tool for visualizing the processes of finite state automata generation and conversion. Compare with the traditional way of teaching, the keyword is interactivity.

Traditional automata theory courses spent significant amounts of time on analyzing algorithms of finite state automata, for instance creation from regular expression or conversion from NFA to DFA, etc. With each example, students do the prescribed steps until they reach the right result. There are some handicaps in this approach. Students who lack real experiences with FSA will have only a shallow sense of the meaning of the analysis. The problems with traditional teaching approach can be divided into two categories: educational weakness in the theoretical approach and experimental weakness due to the fact that students have not worked with the course topics for sufficient time.

Interactive courses enriched with interactive visualization and animation may help students to understand the matter in shorter time with less effort. Also these interactive visualization tools can be used both in teaching and learning.

Herein, GaniFA NG is a very usable tool to assist in teaching, experimenting, and learning about grammars and FSAs, a concept that some students will find particularly challenging. It is the belief of the author that the capabilities described in the fourth chapter will help students to better understand automata theory. Teachers and students will find this tool to be easy to learn and use, allowing the focus to be on the concept. Many students feel that the ability to use their own test data in visualization algorithms would be helpful.

Users can stop step or go to the end of visualization by clicking only one button if he/she wishes. This allows them to stop the visualization algorithm, recheck what has happened so far, and then step forward until they reach the final state or the end of input data. It is much easier to understand and analyze the process since the GaniFA NG tool shows the visualization step by step and with the color assistance of animation steps, and it is much funnier also since the users (or learners) can interact with the computer.

A text box at the top menu located under zooming buttons provides the current status of input by painting it with a convenient color and alerting the user to what has happened and what is going to happen next.

GaniFA NG uses a tree-based layout and this restricts the count and the optimal locations of visual elements in case of using a complex REG. A scroll mechanism works for scrolling the whole animated FSA, but yet the structure of the automaton may be incomprehensible sometimes as in Figure 7.1.

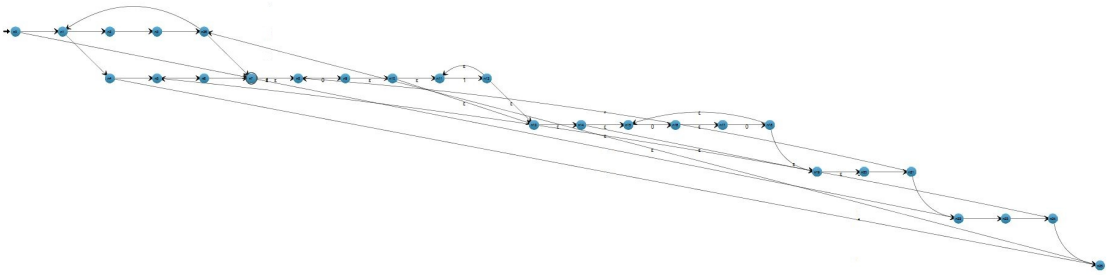


Figure 7.1: Generated NFA with the REG (0|(1(01*(00)*0)*1)*)*

The NFA to DFA conversion reduces the node count in a FSA so that the confusion (if any) becomes less like in Figure 7.2. Nevertheless, a manual arrangement can be needed like in Figure 7.3.

To sum up, GaniFA NG can be used as a tool for the imposing graphical depiction, editing of FSAs and also manipulation of them with user friendly facilities commodiously and provides a complementary alternative to currently available tools focusing on students for whom the subject matter of FSAs is challenging or intimidating.

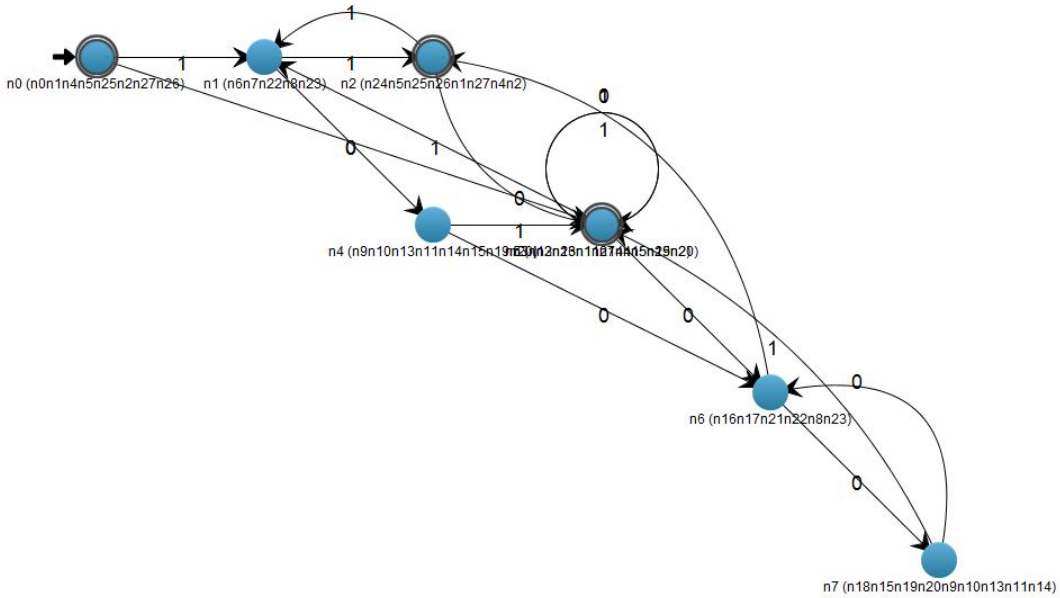


Figure 7.2: The DFA of the generated NFA with the REG (0|(1(01*(00)*0)*1)*)*

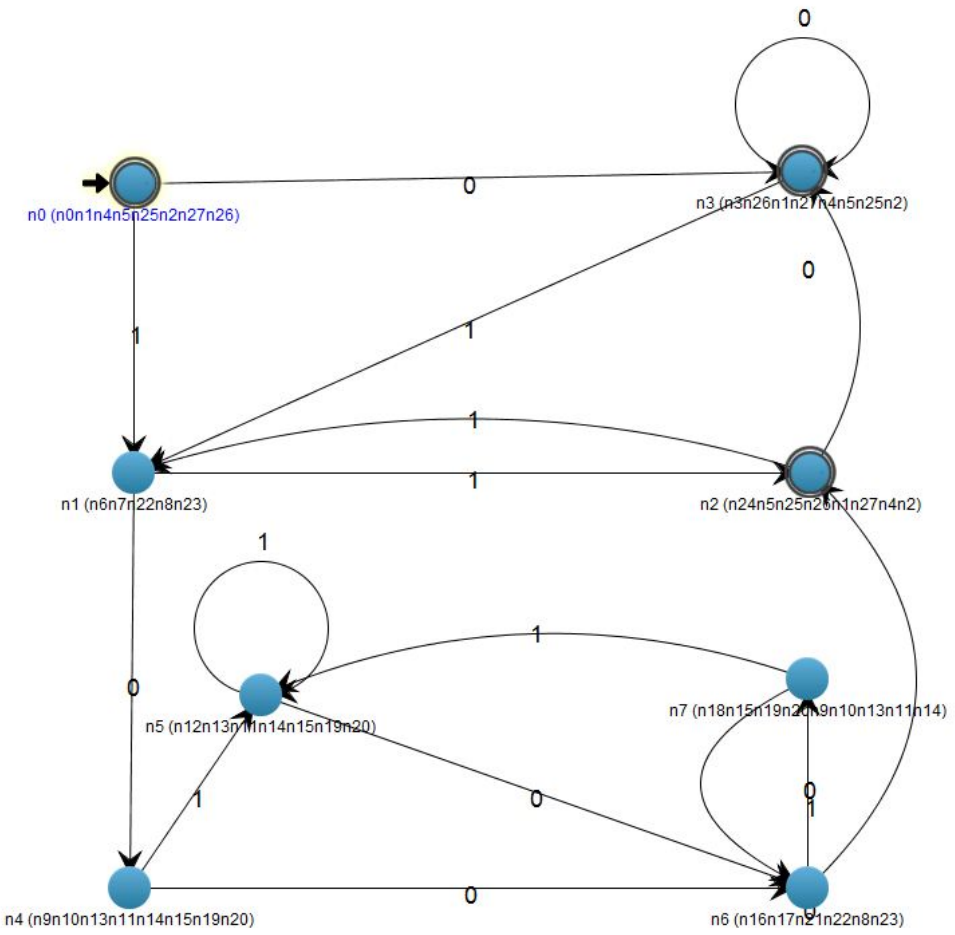


Figure 7.3: The DFA of the generated NFA with the REG (0|(1(01*(00)*0)*1)*)* after the user organizes the DFA layout manually

7.2 Future work

A number of issues are progressed in this thesis, and some of those issues are out of the scope of this thesis, others are our ideas that we got during the research, but there was not much time to implement and research further.

This thesis should not be only design automation software, but it should be an advance skillful research tool and educational system which students will be able to improve for their automata theory knowledge most efficiently and quickly. In future, a few capabilities are expected to be done.

GaniFA NG allows users to create a NFA in two ways. A NFA can be generated by a REG conversion or mouse editing in NFA panel edit mode. In both, these creation process are controlled by an algorithm in the system. However, if a user loads a manipulated FSA, that GraphML file should be checked and approved.

The main scenario is currently proceeding with REG conversion to NFA, NFA to DFA conversion and DFA minimization. A reverse scenario could be developed so that a REG can be acquired from an NFA or a DFA.

An undo and redo mechanisms are potential features can be included in both automaton editing and visualization scenarios in the future. Moreover, new import-export file formats can be supported, and node or edge appearances can be made

modifiable in the future releases. At present, visualization operation can be stopped, stepped or completed immediately. A regress button may be useful for repeating the visualization steps.

Video exporting may be a good practice for produced FSA for those who want to watch what he/she had done before in a later time.

Last but not least, the old GaniFA tool had been tested by more than 100 test persons so that an evaluation could be carried out in order to prove that persons learning with the old GaniFA had a better learning performance than persons learning with other teaching methods. The evaluation gave affirmative results on the subject of its usability and effectiveness. The GaniFA NG also can be considered in the same category as the old GaniFA since it is designed and developed with the same methodology. When viewed from this aspect, it is expected to have an equal or a better test performance in terms of usefulness and beneficialness. Even so, an evaluation process should be determined as a forward looking.

7.3 Social Aspects

There are a large variety of possible applications which are benefited from FSAs: number theory, distributed systems, algorithms on strings, theory of codes, complexity of boolean circuits and others. A good understanding of FSAs may help a software developer in numerous fields. From this point of view, this thesis attempts to provide a visualization tool which can be applied in education of theoretical automata theory in computer science courses. Students usually have difficulties to perceive dynamic processes such as the creation of a FSA, the working of an algorithm or the flow of information during computation. GaniFA NG was developed in order to visualize concrete examples that help the learner to better understand principles of creation and conversion of FSAs. The educational aim of this work is to provide a practical experience and self-efficacy to students.

References

[1] André Almeida, Nelma Moreira, and Rogério Reis. Guitar and fagoo: Graphical interface for automata visualization, editing, and interaction ⋆. Online; accessed 2014-06-01.

[2] H. Bergströms. Applications, minimizations and visualization of finite state machines, master's thesis, stockholm university, 1998. http://people.dsv.su.se/∼henrikbe/petc/. Online; accessed 2014-04-01.

[3] Jean Bovet. Visual automata simulator, a tool for simulating automata and turing machines. http://www.cs.usfca.edu/∼jbovet/vas.html, 2004-2006. Online; accessed 2014-04-01.

[4] Stephan Diehl, Carsten Görg, and Andreas Kerren. Preserving the mental map using foresighted layout. In DavidS. Ebert, JeanM. Favre, and Ronald Peikert, editors, *Data Visualization 2001*, Eurographics, pages 175–184. Springer Vienna, Ascona, Switzerland, 2001.

[5] Stephan Diehl and Andreas Kerren. Increasing explorativity by generation. In *In Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications, EDMEDIA-2000. AACE*, Montreal, Canada, 2000.

[6] Stephan Diehl, Andreas Kerren, and Torsten Weller. Ganifa. http://rw4.cs.uni-sb.de/projects/ganimal/GANIFA/, 1999-2001. Online; accessed 2014-04-01.

[7] Stephan Diehl, Andreas Kerren, and Torsten Weller. Visual exploration of generation algorithms for finite automata on the web. In *Revised Papers from the 5th International Conference on Implementation and Application of Automata*, CIAA '00, pages 327–328, London, UK, UK, 2001. Springer-Verlag.

[8] Danyel Fisher, Joshua O'madadhain, Padhraic Smyth, Scott White, and Yan-Biao Boey. Analysis and Visualization of Network Data using JUNG. *Journal of Statistical Software*.

[9] Eileen F. S. Head. Assist: A simple simulator for state transitions. http://www.cs.binghamton.edu/∼software/ASSIST.html, 1998. Online; accessed 2014-06-01.

[10] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. Introduction to automata theory, languages and computation. 2007.

[11] M. J. Kearns and U. V. Vazirani. An introduction to computational learning theory. 1994.

[12] Andreas Kerren. Learning by generation in computer science education. *Journal of Computer Science & Technology*, 4(2):84 – 90, 8 2004.

[13] Andreas Kerren, Tomasz Müldner, and Elhadi Shakshuki. Novel algorithm explanation techniques for improving algorithm teaching. In *Proceedings of the 2006 ACM Symposium on Software Visualization*, SoftVis '06, pages 175–176, New York, NY, USA, 2006. ACM.

[14] Andreas Kerren and JohnT. Stasko. Chapter 1 algorithm animation. In Stephan Diehl, editor, *Software Visualization*, volume 2269 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin Heidelberg, 2002.

[15] Shiva Kintali. Review of elements of automata theory, by jacques sakarovitch, translator (from french) reuben thomas. pages 45–47, 2012.

[16] M. S. Marshall, I. Herman, and G. Melançon. An object-oriented design for graph visualization. *Software: Practice and Experience*, 31(8):739–756, 2001.

[17] W. Patrick Merryman. Animating the conversion of non-deterministic finite state automata to deterministic finite state automata, master's thesis, montana state university, 2007.

[18] Andrés Moreno. Algorithm animation. In Andreas Kerren, Achim Ebert, and Jörg Meyer, editors, *Human-Centered Visualization Environments*, volume 4417 of *Lecture Notes in Computer Science*, pages 295–309. Springer, 2006.

[19] Ok-Choon Park and Reginald Hopkins. Instructional conditions for using dynamic visual displays: a review. *Instructional Science*, 21(6):427–449, 1992.

[20] Jean-Éric Pin. Languages and automata. In *Mathematical Foundations of Automata Theory*, pages 40–56, 2014.

[21] Guido Rößling, Mike Joy, Andrés Moreno, Atanas Radenski, Lauri Malmi, Andreas Kerren, Thomas Naps, Rockford J. Ross, Michael Clancy, Ari Korhonen, Rainer Oechsle, and J. Ángel Velázquez Iturbide. Enhancing learning management systems to better support computer science education. *SIGCSE Bull.*, 40(4):142–166, November 2008.

[22] Guido Rößling, Thomas Naps, Mark S. Hall, Ville Karavirta, Andreas Kerren, Charles Leska, Andrés Moreno, Rainer Oechsle, Susan H. Rodger, Jaime Urquiza-Fuentes, and J. Ángel Velázquez-Iturbide. Merging interactive visualizations with hypertextbooks and course management. *SIGCSE Bull.*, 38(4):166–181, June 2006.

[23] Susan H. Rodger, Anna O. Bilska, Kenneth H. Leider, Magdalena Procopiuc, Octavian Procopiuc, Jason R. Salemme, and Edwin Tsang. A collection of tools for making automata theory and formal languages come alive. *SIGCSE Bull.*, 29(1):15–19, March 1997.

[24] GraphML Team. The graphml file format. http://graphml.graphdrawing.org/, 2002. Online; accessed 2014-06-01.

[25] The JUNG Framework Development Team. Jung, java universal network/graph framework. http://jung.sourceforge.net/index.html, 01 2010. Online; accessed 2014-04-01.

[26] John von Neumann. Collected works; vol. 5: Design of computers, theory of automata and numerical analysis. pages 290–326, 1963.

[27] Timothy M. White and Thomas P. Way. jfast: A java finite automata simulator. *SIGCSE Bull.*, 38(1):384–388, March 2006.

Lnu.se

Linnæus University
Sweden

Faculty of Technology
SE-391 82 Kalmar | SE-351 95 Växjö
Phone +46 (0)772-28 80 00
teknik@lnu.se
Lnu.se/faculty-of-technology?l=en