

VISUALIZING THE MINIMIZATION OF A DETERMINISTIC  
FINITE STATE AUTOMATON

by

Kshatriya Jagannath Rajini Singh

A thesis submitted in partial fulfillment  
of the requirements for the degree

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY  
Bozeman, Montana

November 2007

©COPYRIGHT

by

Kshatriya Jagannath Rajini Singh

2007

All Rights Reserved

APPROVAL

of a thesis submitted by

Kshatriya Jagannath Rajini Singh

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the Division of Graduate Education.

Dr. Rockford. J. Ross

Approved for the Department of Computer Science

Dr. John Paxton

Approved for the Division of Graduate Education

Dr. Carl A. Fox

## STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library.

If I have indicated my intention to copyright this thesis by including a copyright notice page, copying is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U. S. Copyright Law. Requests for permission for extended quotation from or reproduction of this thesis in whole or in parts may be granted only by the copyright holder.

Kshatriya Jagannath Rajini Singh

November 2007

## ACKNOWLEDGEMENTS

I would like to thank my academic advisor Dr. Rockford. J. Ross for giving me the opportunity to do this work, for his guidance and patience throughout my Masters program. Dr. Ross has been a very strong mentor to me these past few years and I have learned immensely from him not only technically regarding this work, but how to approach work and research in general. His mentoring will influence me far beyond this work. His time and guidance are truly appreciated.

I would also like to thank Dr. Denbigh Starkey and Prof. Lloyd for their guidance, encouragement, and for serving on my thesis committee. I am also thankful to the CS department staff Jeannette Radcliffe and Kathryn Hollenback, who have always made the environment congenial and have helped me in every possible way.

I would like to thank God and my family who are my pillars of strength and encouragement, including my parents Mr. Jagannath Singh and Mrs. Nalini Singh, and my brothers Giridhar Singh and Srinath Singh. Their contributions cannot be put in words and i cannot thank them enough. And last but farthest from least, i would like to thank my invaluable friends.

## TABLE OF CONTENTS

1. INTRODUCTION .....	1
2. THESIS .....	3
3. RELATED AND PREVIOUS WORK.....	5
Webworks Laboratory Projects.....	5
Context Free Grammar Animator .....	6
FSA Animator .....	7
Pumping Lemma Animator .....	13
Regular Expression Animator .....	14
Pushdown Automata .....	16
NFA to DFA conversion Animator.....	18
JFLAP .....	20
Minimizing a DFA .....	21
4. VISUALIZING THE MINIMIZATION OF DETERMINISTIC FINITE STATE AUTOMATON.....	26
Background .....	26
Equivalent States .....	27
Distinguishable States .....	27
The Algorithm .....	27
Visualizing the algorithm .....	29
Step 1 – removing useless states .....	29
Step 2 – identifying final and non final states .....	29
Step 3 – initial marking of state pairs .....	30
Step 4 – determining distinct states .....	31
Step 5 – completing the marking of states .....	31
Step 6 – merging the redundant states .....	31
Step 7 – reforming states and transitions .....	31
Step 8 – resetting the start state .....	32
Step 9 – removing individual states that were merged .....	32
5. DFA MINIMIZATION APPLET .....	33
MINIMIZE DEMO VISUALIZATION .....	34
6. CONCLUSIONS.....	49
7. EVALUATION .....	50
The Regular Expression Animator.....	50
Evaluating the DFA Minimization Applet .....	51

## TABLE OF CONTENTS (CONTINUED)

8. FUTURE DIRECTIONS .....	53
Exercise Mode.....	53
Minor Extensions .....	53
BIBLIOGRAPHY .....	55
APPENDICES.....	57
APPENDIX A – SAMPLE PRE AND POST TEST.....	58
SAMPLE QUESTIONS FOR PRE.....	59
AND POST TEST.....	59
APPENDIX B – ASSIGNMENT SAMPLE.....	61
SAMPLE ASSIGNMENT QUESTIONS.....	62
APPENDIX C – FORMATIVE FEEDBACK QUESTIONNAIRE .....	63
SAMPLE FORMATIVE .....	64
FEEDBACK QUESTIONNAIRE .....	64

## LIST OF FIGURES

Figure	Page
1. Lutey's Context Free Grammar Animator- LL(1) Animator .....	7
2. Grinder's FSA .....	8
3. Grinder's FSA in Run mode .....	10
4. FSA in Compare mode displaying a string that is not part of the language .....	11
5. A correct FSA .....	12
6. Cogliati's Pumping Lemma in one of the four modes .....	13
7. Walsh's Regular Expression Animator .....	14
8. William's Pushdown Automata .....	16
9. FSA in Demo mode .....	18
10. Rodger's JFLAP- Minimization of DFA .....	22
11. Rodger's JFLAP- Minimization of DFA with the pop up box to enter a terminal .....	23
12. Start of Minimization demo .....	34
13. Applet showing useless states in red .....	36
14. Applet showing non final states .....	37
15. Applet showing the first pair of states being marked .....	38
16. Applet showing pairs of distinct states .....	39
17. Applet displaying one of the pairs of states that can be merged .....	40
18. Applet displaying all pairs of states that can be merged .....	41



## LIST OF FIGURES (CONTINUED)

Figure	Page
19. Applet displaying merged states .....	42
20. Applet displaying transitions within the merged states .....	43
21. Applet displaying transitions among other states and the merged states	44
22. Applet cleaning up transitions among individual states that form merged states and other states .....	45
23. Applet displaying the new start state .....	46
24. Applet with individual states which are now part of the merged states, being removed .....	47
25. Applet displaying the minimized DFA .....	48
26. Sample DFA .....	60

## ABSTRACT

The minimization of deterministic finite state automata is one of the challenging concepts that students of an introductory theory of computation course must master. The minimization algorithm identifies redundant states that may occur in any deterministic finite state automaton and combines them in order to minimize the total number of states. Traditional textbooks provide a static presentation of this process. The average student seems to grasp the process better if it is presented dynamically.

In this thesis we visualize every step of a minimization algorithm in sequence through a Java applet. Animations are used at every step in order to bring out the dynamism of the process. Students are also presented with a demonstration of both the original automata and the minimized automata running on the same input, in order to see that both automata accept/reject the same language.

The applet will be included in a hypertextbook project as a benefit to both the instructor and the student.

## INTRODUCTION

Minimization of a deterministic finite state automaton is one of the challenging concepts for theory students at an introductory level to understand and learn. The focus of this thesis is the development of a visual software system that aids student understanding of the minimization process and the formulation of a strategy for evaluating the efficacy of the developed system on student learning. The idea is that by a dynamic and a detailed approach to learning, students will be able to understand the minimization process better and be able to apply it on their own.

This visual software system is intended to be incorporated into a much larger system called a hypertextbook [2]. Ross, who is the author of some hypertextbooks, describes a hypertextbook this way

The hypertextbook is a novel teaching and learning resource built around web technologies that incorporates text, sound, pictures, illustrations, slide shows, video clips, and most importantly, active learning models of the key concepts of the theory of computing, into a single integrated source.

The hypertextbook is thus a complete learning resource that provides not only text presentations, but also includes representations of important concepts through active learning applets that can be accessed through any standard web browser, and can be used by both instructors and students. *Theory of Computing: The Hypertextbook* is proposed to be the first working example of the hypertextbook concept. The minimization of deterministic finite state automaton software of this thesis will become part of this larger system.

This theory hypertextbook is the result of the combined efforts of a group of people with the intent of presenting the dynamism of the concepts of the theory of computation through visualizations. It contains a collection of several active learning applets covering different aspects of the theory of computation, such as finite state automata, regular expressions, pushdown automata, and others. The hypertextbook has been designed and developed with the intent of augmenting or replacing the traditional static textbook paradigm of teaching and learning with that of the animated versions, which have been more effective in helping students learn.

This approach of using active learning applets will be simple enough for instructors to integrate into their curriculum and it will free them to focus on the underlying theory rather than on preparing intuitive presentations of the models of computations. Along with being useful in the classroom, the applets of the hypertextbook help students study on their own. They will not have to restrict themselves with the number of examples that they get to deal with in the class. Instead, they can go over other examples that were not covered by the instructor, on their own, thereby increasing the depth of their understanding of the concept.

## THESIS

The tenet of this thesis is that it is possible to design and construct an applet that demonstrates the minimization of a finite state automaton that helps students learn this process at least as well as they would if they used a traditional textbook instead. The major effort of this thesis was to build an applet that demonstrates the algorithm of minimizing a deterministic finite state automaton (DFA). The applet has been designed and developed with average students in mind. It guides them through the conversion process at a deliberate pace, one step after another. The applet encourages both instructors and students to construct their own examples by providing them with directions for using the controls to create and label the states and transitions.

Measuring the effectiveness of the above applet was also a consideration. This can best be done in a classroom environment consisting of two groups, one being the test set and the other being the control set. The test set would learn the concept by having access to the applet and by experimenting with its different examples. The control set would learn the concept without having access to the applet.

The applet was successfully constructed and is fully functional. A measure of its performance could not be completely done as it was not completed at the time when the theory class was in progress. However a few preliminary questionnaires were given to the students of the previous offering of that course to evaluate the effectiveness

of active learning applets in general. These helped guide the development of this project. Also, a set of instruments for evaluating the applet was developed for use in future offerings of the theory course. These appear in appendices A and B.

## RELATED AND PREVIOUS WORK

This section discusses the fundamental work done on the FSA applet and the previous work done on the process of minimizing a DFA.

### Webworks Laboratory Projects

The Webworks Project [1, 2] at Montana State University is the result of a collection of various projects put together with a focus on developing a complete hypertextbook on the theory of computing. The aim of this hypertextbook is to serve as a single integrated theory learning resource based around active learning models of the key concepts in a theory of computation course. The hypertextbook is accessible through any standard web browser. The project is done partly with the intention of minimizing the effort required of instructors in finding and integrating web-based animations of key theory concepts into their courses. A fully integrated resource will also help clear the misconception of students that web-based animations lead to extra unnecessary work [3]. The embedded animations will also help students construct correct mental models of important theory concepts.

Mental models are the small-scale representations of real or imaginary situations that the human mind constructs and uses to anticipate events [4]. They can be constructed from perception/observation, imagination, or the comprehension of discourse [5]. Hence, in order to construct a functionally correct mental model of a particular

concept, students must be given a chance to observe and model the behavior that leads to positive reinforcement [6].

The algorithm of minimizing a DFA central to this thesis will help students construct an accurate mental model of the minimization process. The applet will be integrated into the theory hypertextbook of Dr. Ross, although it can also be used as a standalone system.

### Context Free Grammar Animator

A context free grammar animator was developed by Teresa Lutey [1]. It involves animating the parse trees that represent the expansion of the strings belonging to the context free grammar.

In order to run the animator, the user has options to select a grammar from the library provided or to supply one of their own. The user can select any nonterminal for expansion and a rule to use for that expansion. This expansion of the string belonging to the grammar is represented by parse trees. Figure 1 is a snapshot of this animator in action.

There are many other tools and animation applets that have been built and included in the hypertextbook. Most of them are being applied in the classroom by Dr. Ross as part of the compilers course. As these were not used as the basis for this thesis, they have been excluded from any description here.



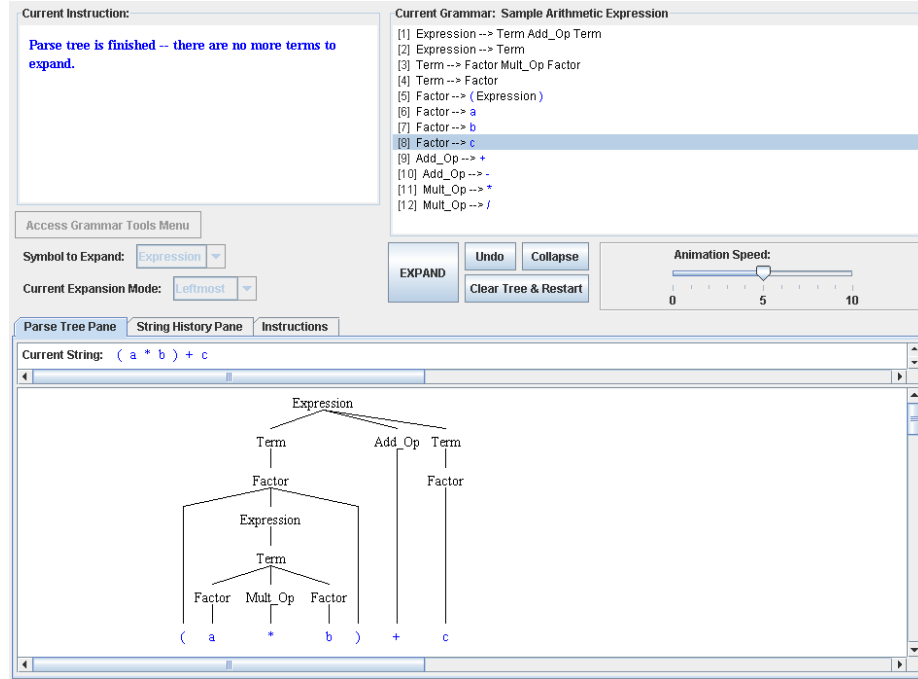


Figure 1. Lutey's Context Free Grammar Animator-LL(1) Animator.

## FSA Animator

The Finite State Automata Animator (FSA) upon which the applet of this thesis is based, was created by Michael Grinder [7, 8, 9] and forms the basis for the minimization animator of this thesis. The FSA animator is designed to be a visualization tool that animates the operation of an FSA.

The idea behind building this FSA Java applet was to animate the operation of an FSA and to animate the process of running the FSA on various input strings. The FSA applet also has an exercise feature that provides the students with the flexibility of creating their own FSA's and running various input strings on them. This feature compares the FSA provided by the students with a background hidden

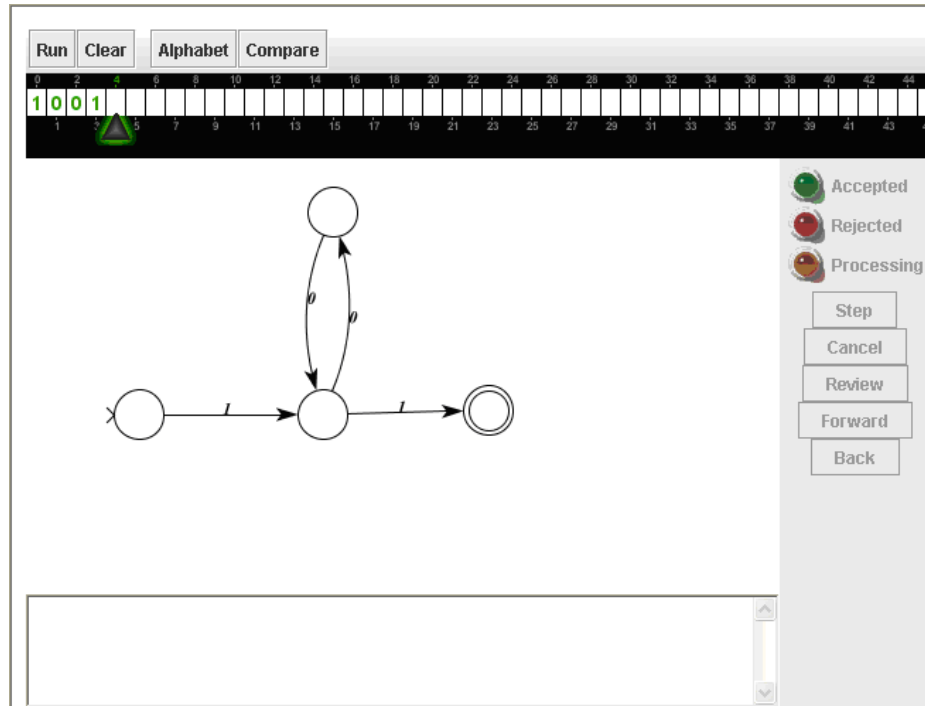


Figure 2. Grinder's FSA.

FSA which is the correct FSA given by the instructor. Therefore, the student's FSA would be termed correct only if it recognizes the language that the hidden correct FSA recognizes. This FSA applet is designed in such a way that it works for both deterministic FSA's and non-deterministic FSA's.

In the FSA applet an FSA can be constructed by using mouse clicks combined with the control key. States can be created and moved around as desired. Transitions can be created by using the same combination of a mouse click with control key and by dragging it from the states from which a transition starts and ending it at the state where the transition ends.

As seen in figure 2, the FSA animator applet displays a series of buttons that can be clicked at its very top. The *Run* button engages the automaton into processing mode; this makes the automaton read the symbols that are available on the virtual tape one at a time and, depending on the symbol being read and the action defined, changes the state of the automaton. The virtual tape that appears at the top of the applet and just below the buttons is made up of a grid of cells, holding the string that is to be processed by the automaton. The FSA tape head is represented by a triangle that appears at the bottom of the tape cells.

The *Clear* button pops up a dialog box with options to either clear the tape, clear the alphabet, or to clear the FSA. Depending on the choice opted, the corresponding action is taken. The *Alphabet* button displays the alphabet dialog box that contains a variety of symbols that can be used to build an FSA. Symbols can be selected individually. And once all the symbols are selected, clicking on the *OK* button on the alphabet dialog box confirms all the symbols as being used for the FSA under consideration, and closes the dialog box.

As seen in figure 3, the FSA is in the *Run* mode. A yellow light is turned on which signifies that the FSA is currently processing the input string available on the virtual tape. The *Step* and the *Cancel* buttons are also now enabled. The *Review* button activates the *Forward* and the *Back* buttons which allow the user to move forward or back through the currently processed tape.

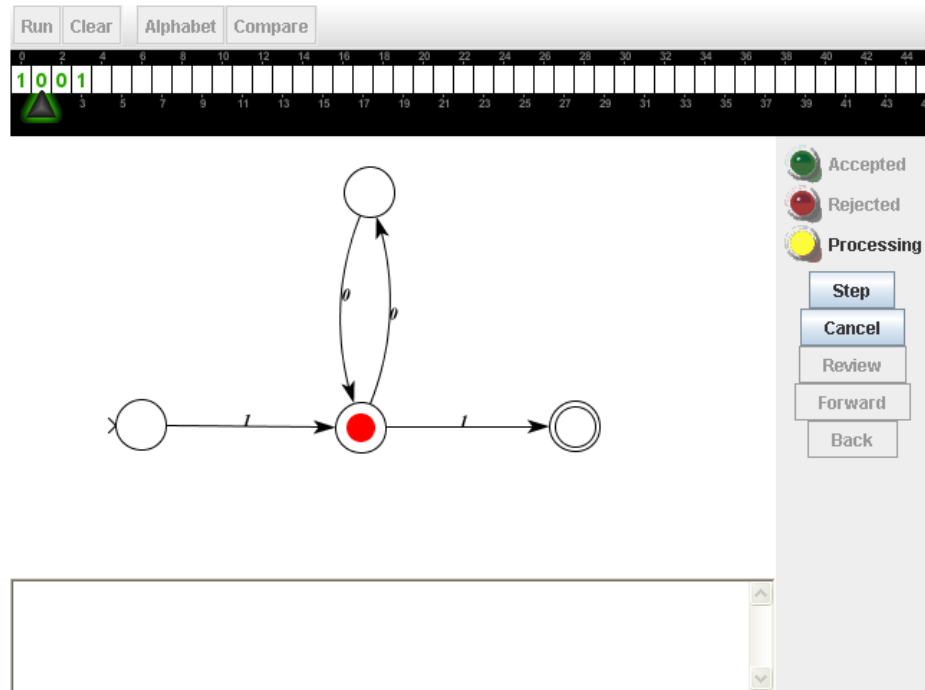


Figure 3. Grinder's FSA in Run mode.

A red circle moves along the transition arrows between states of the FSA on every click of the *Step* button, effectively demonstrating the effect of processing the current input symbol. Acceptance or rejection of an input string is determined by the state in which the red circle is after reading the entire string. If the red circle is in a final state the circle turns green, the yellow light from the panel of lights is turned off, and a green light is turned on. Otherwise the red circle remains red, the yellow light is turned off, and a red light is turned on.

The *Compare* button adds to the exercise feature of the FSA applet. This way instructors can develop exercises for the students to work on and provide a correct FSA hidden in the background. Students can then create their own FSA according to

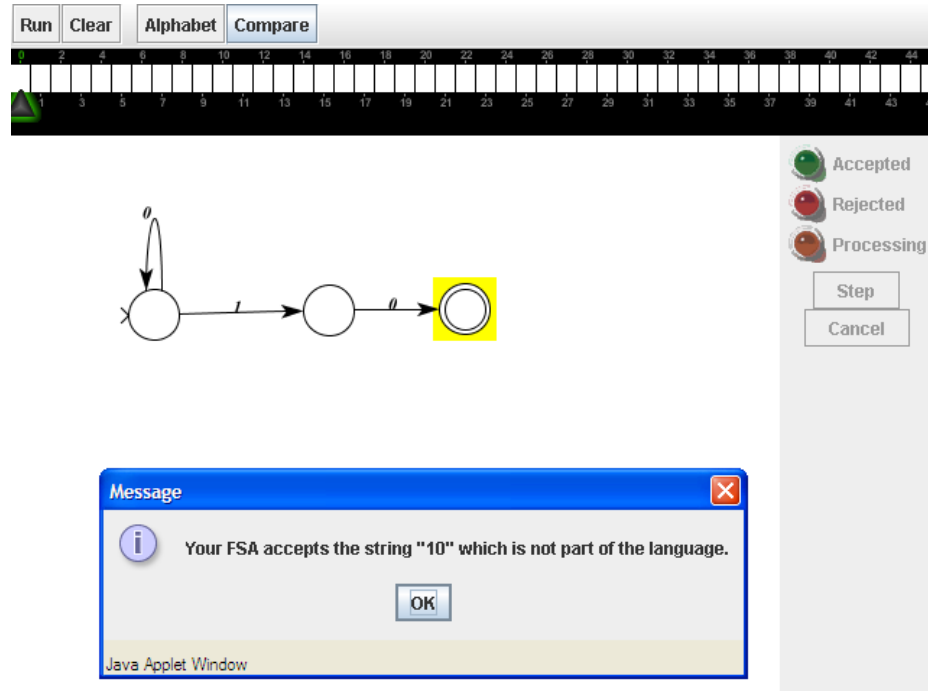


Figure 4. FSA in Compare mode displaying a string that is not part of the language.

the exercise and measure its correctness during the construction process by clicking on the *Compare* button to compare their FSA against the hidden, correct automaton in the background. If the automaton constructed is not correct, messages showing example strings on which the automaton's language acceptance differs from that of the hidden correct automaton are displayed.

Figure 4 emphasizes the *Compare* feature and shows the pop up that displays a string that is not part of the language of the FSA currently constructed.

Figure 5 shows a possible correct FSA. The process of building an FSA is quite simple. An individual state is created by pressing and holding the Ctrl key while left clicking with the mouse on the exact location where it is desired to have a state. If at

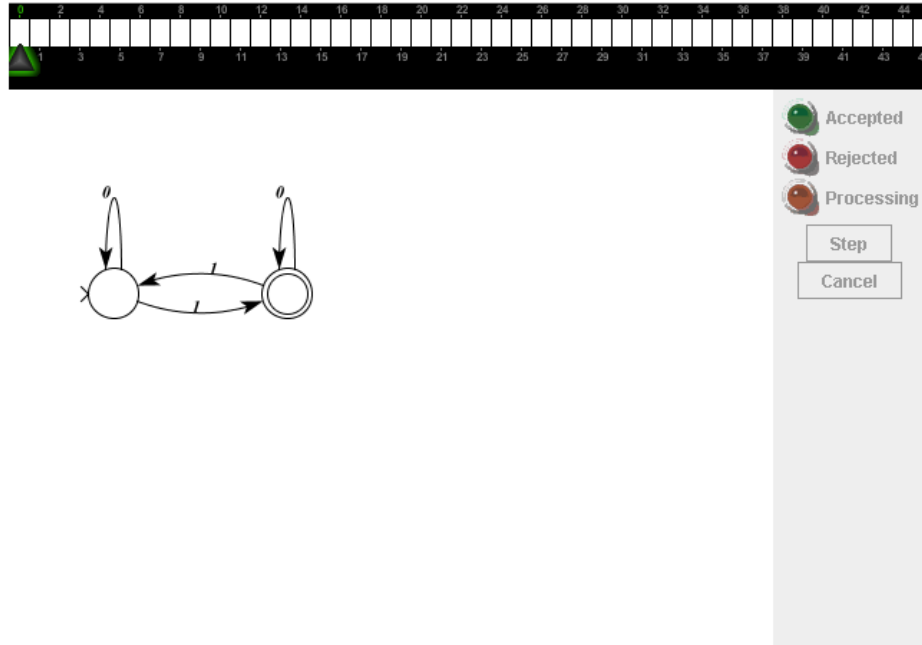


Figure 5. A correct FSA.

a later stage, it is required to have a particular state at a different location, this can be accomplished with a left mouse click over the state and dragging and dropping it at the location desired. A state can be labelled and can be set as start/final state by placing the mouse over the state and right clicking over it. Transitions are created by clicking inside the state from which a transition is desired, and dragging and dropping the transition arrow inside the state where it has to end. Right clicking on the transition while it is highlighted allows the user to select the alphabet symbols governing the transition.

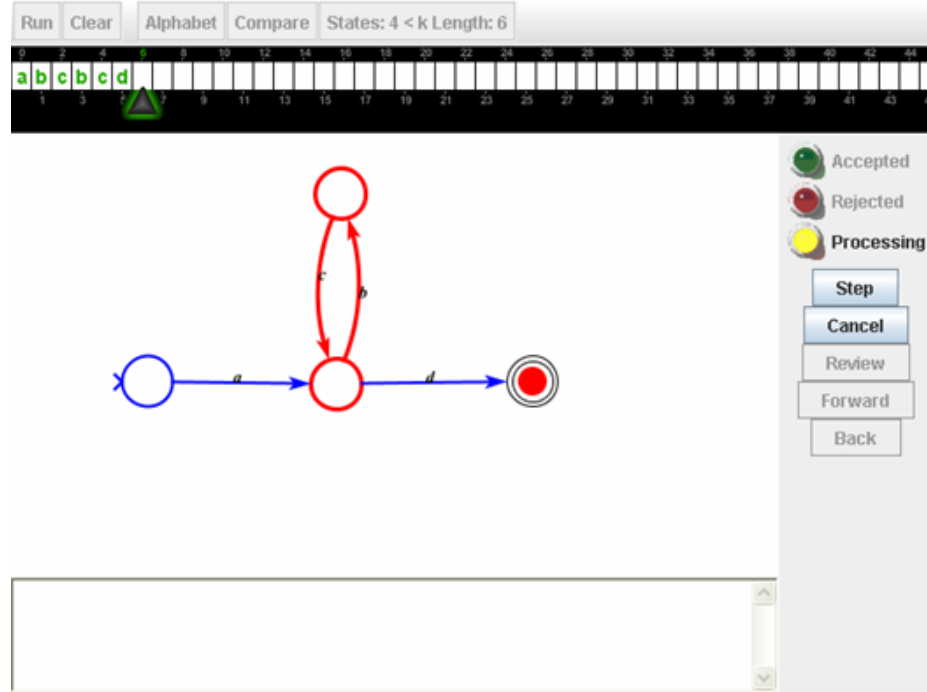


Figure 6. Cogliati's Pumping Lemma in one of the four modes.

### Pumping Lemma Animator

Josh Cogliati [1] added four more modes to Grinder's [7, 8, 9] FSA to animate the pumping lemma for regular languages. The visualization is made effective by breaking the input string into 3 parts— $x$ ,  $y$  and  $z$ . The first mode illustrates a way of dividing an input string into 3 separate sections—section  $x$ , section  $y$  and section  $z$  according to the pumping lemma—such that pumping  $y$  will result in new strings belonging to the language. The second mode provides the capability to the user to select a subsection of the string  $w$  to break into  $x$ ,  $y$  and  $z$  sections.

The third mode again allows the user to select a subsection of string  $w$ , but the subsection should be such that it causes the FSA to loop. The applet then checks to

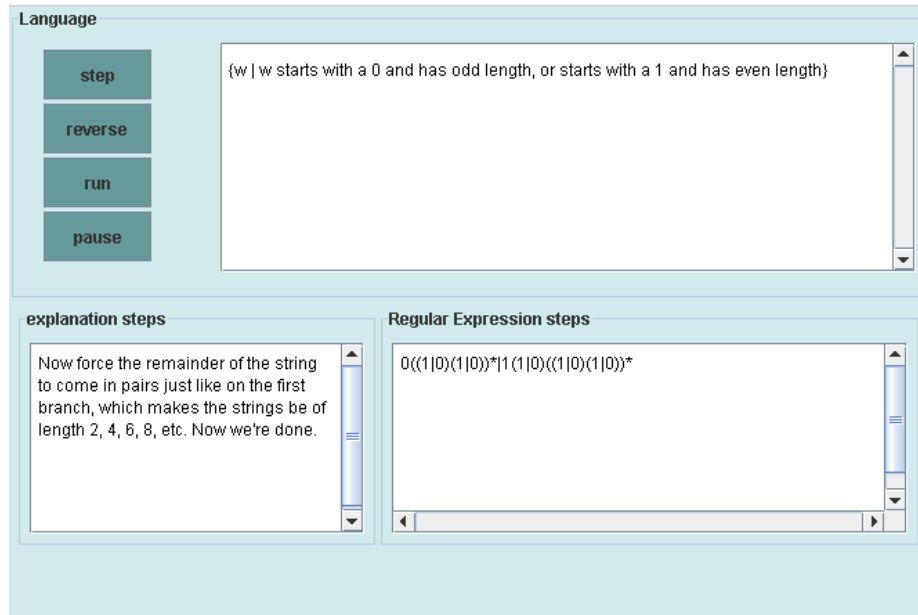


Figure 7. Walsh's Regular Expression Animator.

see that the selection was correct or not. The fourth mode highlights the states and the transitions that have already been visited as part of processing the input string. Figure 6 display's the fourth mode with states and the transitions highlighted.

### Regular Expression Animator

A regular expression animator was developed by Katie Walsh and was later extended by Brad Pascoe [1]. The visualization applet is divided into 3 panes with a toolbar containing buttons on its left side. It has a demonstration and an exercise mode. In the demonstration mode, the user is given a language and is shown how the regular expression that is responsible for it can be built. The topmost pane in



the applet contains a description of the language being considered. The toolbar has *Step*, *Reverse*, *Run*, and *Pause* buttons.

Figure 7 shows the regular expression animator in demonstration mode. On every click of the *Step* button the lower left pane provides the user with an explanation about the process taking place, and the the lower right pane displays the regular expression that is being built. Clicking the *Reverse* button will back up the presentation by one step. The *Run* button starts the applet running through the entire animation with short pauses at each step. The *Pause* button pauses the run mode.

The exercise mode requires the user to construct a regular expression for a given language and provides an auxiliary regular expression to use in their solution. Feedback is also provided on the correctness of the user proposed solution.

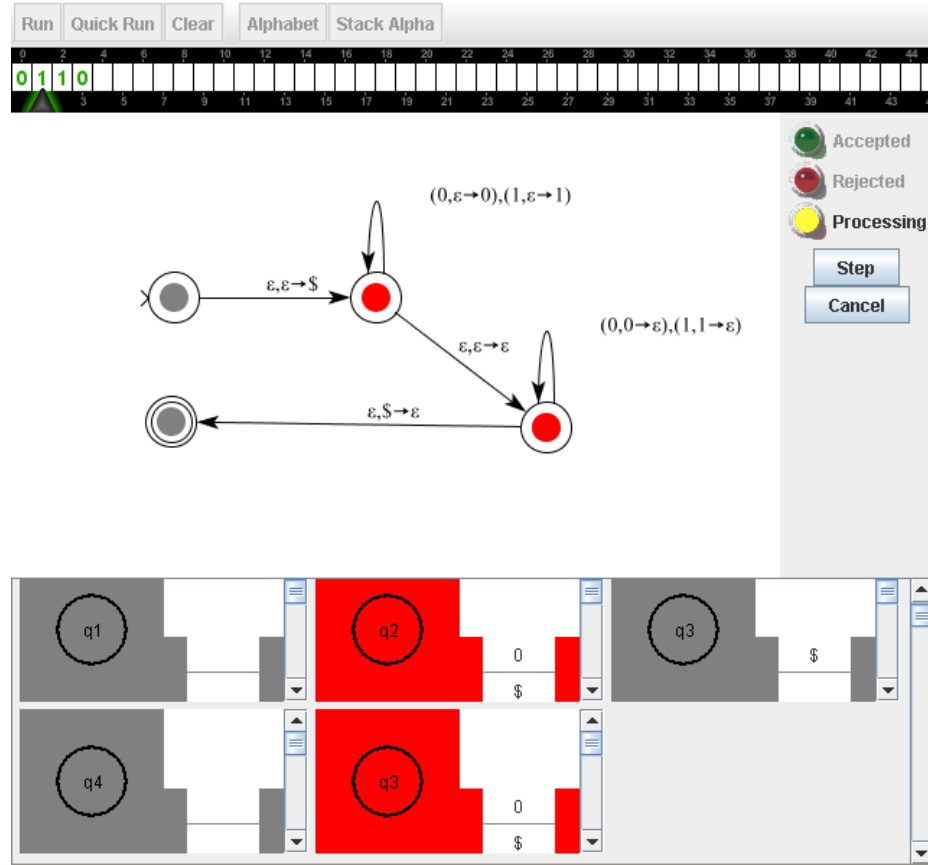


Figure 8. William's Pushdown Automata.

### Pushdown Automata

Pushdown automata were visualized by Cheston Williams. In his applet there were two modes for the automaton; an example mode and an exercise mode. In the example mode, the user is required to give an input string that is to be processed by the PDA and the processing of that string is animated. The applet displays the finite state component and the stack of the PDA. If the PDA is in multiple states for a given symbol (as nondeterminism is also allowed in the PDA applet), stacks for each active state are shown. Figure 8 shows a PDA running on the input string 0110.

Different colors are applied to the stacks and active states to signify whether they are active or not. Red signifies that the stack/state is active, grey signals dead, and green signifies that the input was accepted. Dead stacks are the result of unproductive non deterministic branches, and these stacks are removed. In figure 8 the states  $q_1$ ,  $q_3$  and  $q_4$  are active states. State  $q_3$  is both active and dead which is the result of nondeterminism. The state is grey in color initially which is masked by the red dot over it.

In the exercise mode, the applet runs the user constructed PDA on a test suite of valid and invalid strings providing the student with feedback about its correctness. The user constructed PDA is said to be correct if all strings pass the test. This is done in the way described above, as it is theoretically impossible to determine whether or not two arbitrary PDA's recognize the same language.

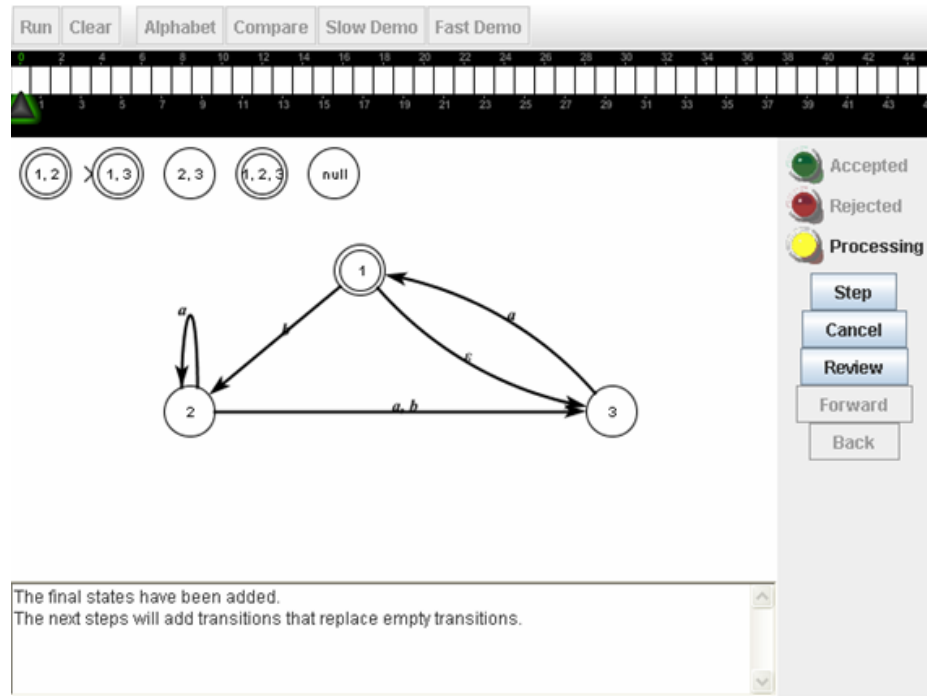


Figure 9. FSA in Demo mode.

### NFA to DFA conversion Animator

A NFA to DFA animator was developed by William Merryman [10] based on Grinder's FSA [7, 8, 9]. The applet has four major modes: the *Detail Demo* mode, the *Demo* mode, the *Active* mode and the *Default* mode. The *Detail Demo* mode places the *Detail Demo* button on the control panel and this button allows the user to view the conversion process at a very slow pace with every step described in great detail.

Figure 9 shows the FSA in *Demo* mode. The *Demo* mode places the *Slow Demo* and *Fast Demo* buttons on the control panel of the applet. The *Fast Demo* mode is intended for users who are familiar with the conversion algorithm, but who wish

to review certain sections, and don't need all the detail that the *Detail Demo* mode provides. The *Active* mode places the *Self-Exercise* button on the control panel of the applet, which allows the user to build a DFA that accepts the same language as the given NFA, using the same steps that the *Demo* and the *Detail* modes use. The *Default* mode places the *Slow Demo*, *Fast Demo* and *Self-Exercise* modes on the control panel.

The idea behind the conversion process is to construct a DFA from the existing NFA, a process that involves four steps. The first step is the construction of new states which essentially are the combinations of the existing states. There is also an additional null state that represents the empty set of states. The second step sets the new start which is the union of the original start state with any state that the start state transits to on an  $\epsilon$  transition. The third step sets the final states as those that contain an original final state in their state set. The fourth step determines the transition function  $\delta'$  for the DFA such that, for a state  $R_1 \in Q'$  and an input  $a$ ,  $\delta'(R_1, a)$  returns  $R_2 \in Q'$ , where  $R_2 = \cup_{r \in R_1} E(\delta(r, a))$ .

JFLAP

The other comprehensive software system that helps students visualize various models/concepts of the theory of computing is Susan Rodger's JFLAP (Java Formal Language Automata Package) [11]. Following are some of the concepts that have been covered by JFLAP:

- Finite state automata
- Pushdown automata
- Turing machines
- Multi-tape turing machines
- Grammars
- L-Systems
- Regular expressions

JFLAP is a good practicing tool for users who are already aware of the concepts and want to be expert at it. The tutorials and help menus are good places for getting the details straight, but there is no explanatory text involved in the applets.

### Minimizing a DFA

Since this thesis concentrates on animating the minimization of a DFA we examine this feature of JFLAP in detail. The first step is to create a DFA by selecting the *State Creator* button to create states. The states are set to the status of initial state or final state by using the *Attributes Editor* button. Transitions for those states can be created by using the *Transition Create* button and then clicking on the state from where the transition should start and dragging and ending it at the state where the transition should end. There are minor difficulties with using JFLAP that can confuse the user. For instance, there is no initial preset of the automaton's alphabet, instead a label editor pops up allowing the user to input label for that transition. An accidental hit of a space bar while entering the label for a transition will cause the applet to accept a space as part of the label.

The next step is to select the *Minimize DFA* option from the *Convert* menu. This separates the applet into two panes where the left hand side has the entire DFA and the right hand side has the current workspace. The current workspace has the DFA represented in a tree structure with the root node being the one with all the states of the DFA. This is then broken down into a left child node and a right child node. The criterion is to separate the final state from non final states, where the left child node represents all the non final states and the right child node represents all the final states. Clicking on any one of these nodes will highlight the states that are represented by that node on the left pane and will also activate some of the tabs that

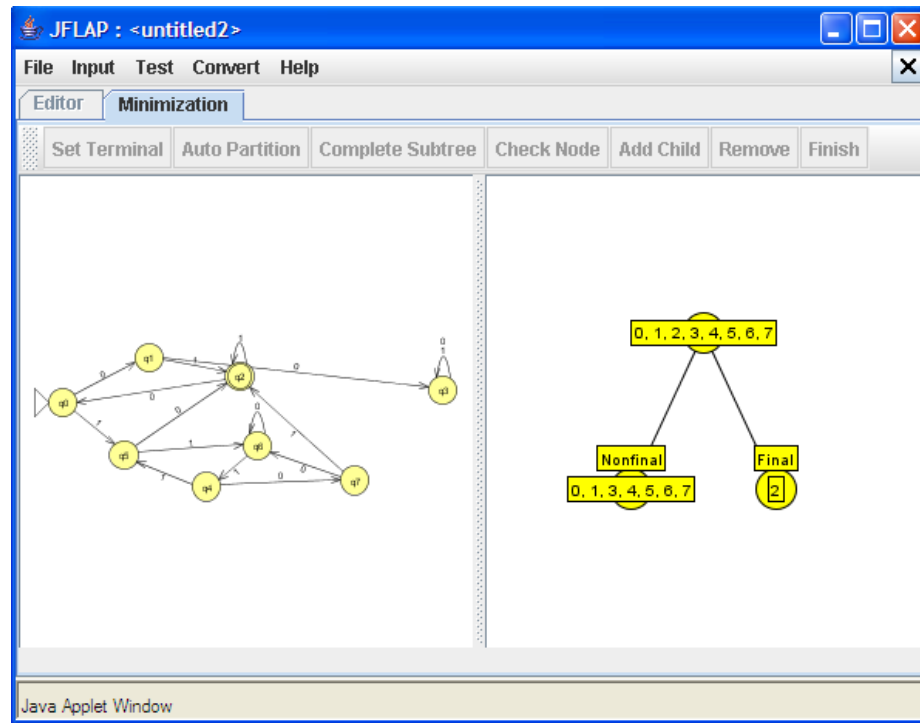


Figure 10. Rodger's JFLAP–Minimization of DFA.

were previously greyed out on the applet. For instance the *Set Terminal* tab that was initially greyed out is now activated as one of the nodes were selected.



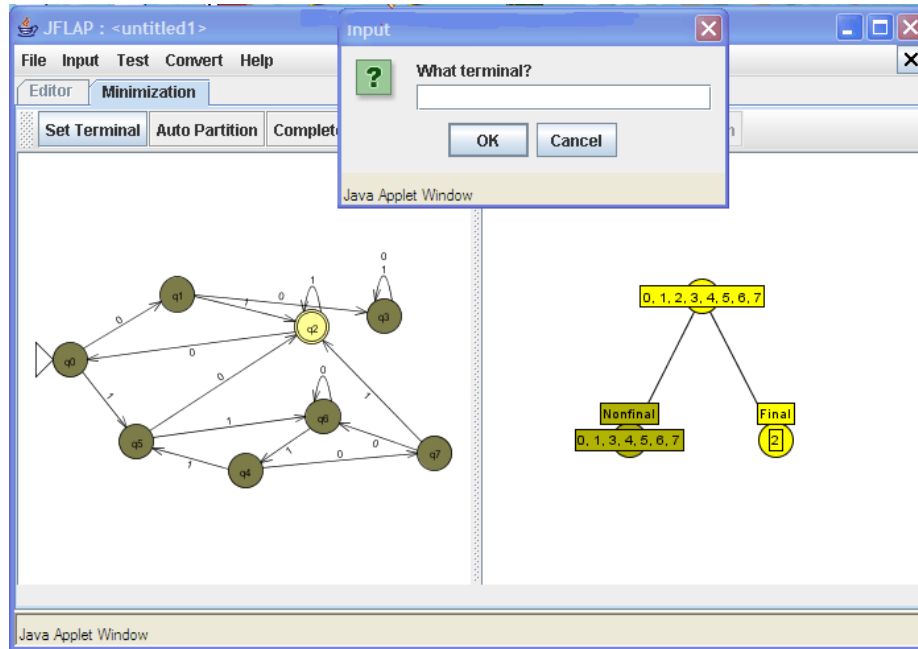


Figure 11. Rodger's JFLAP–Minimization of DFA with the pop up box to enter a terminal.

Clicking on the *Set Terminal* tab will bring up a popup box asking the user to enter a terminal symbol. Depending on the input, which is one of the transition labels, the applet then branches to the group of states in the node. Figure 11 displays the DFA with the pop up box to enter a terminal. If there is no split of the group on that terminal symbol a pop up box appears that gives a warning message about the group not splitting on that terminal symbol. Otherwise the states in the group within a node will split on a terminal symbol into 2 child nodes.

These child nodes can again have a single state or a group of states depending on how many transit to a state within the same group. The group of states is split in the following way; for every state that occurs in the group within a node, a check is made to see whether the state transits on the terminal that the user enters in the pop up box that appears on clicking on the *Set Terminal* tab to the state that is present in the group of the other node (at the same level). If it does transit, then it separates itself from the rest of the states by branching out into a child node that will hold such states. Otherwise, another child node is formed that will hold a group of states that transit to one of the states in the same group.

Every such group of states that transit to one particular state on a terminal symbol is combined into one state. The functionality of *Auto Partition* and the *Complete Subtree* tabs are very similar. They both require the user to select a node. The former only branches out for the currently selected node, whereas the latter displays the complete subtree.

Once two or more states are combined to form a single state, the *Finish* tab gets highlighted. Clicking on this tab will replace the righthand pane with a new pane that contains the combined states. This pane is used to add transitions between the states by using the transition create button. The *Hint Complete* and *Done?* buttons help the user in this process. If the user is finished and the transitions are all correctly added, on hitting the *Done?* button, the minimal DFA should be present in a new window.

Although JFLAP presents the conversion process, it seems not quite detailed and animated enough to be able to replace a textbook. The user has to scroll up and down to follow what exactly is going on inside the applet. Users must not only take the tutorial on how to use the applet, but also understand the concept being covered within the applet. From a naive or intermediate student's prospective this author believes that JFLAP<sup>1</sup> does not provide much help in improving the learning process. Also from an instructor's point of view, a textbook will be needed for the class in addition to JFLAP<sup>2</sup>. Indeed, JFLAP was not designed for use as an independent teaching and learning resource.

---

<sup>1</sup>This issue has also been partly addressed in that JFLAP has been designed in conjunction with the theory textbook by Peter Linz [13].

<sup>2</sup>This issue has been partially addressed through a publication of a JFLAP manual [12].

## VISUALIZING THE MINIMIZATION OF DETERMINISTIC FINITE STATE AUTOMATON

### Background

Finite state automata are best thought of as pattern recognizers. Certain commonly used programs, such as the lexical analyzers found in compilers, are often designed and built as finite state automata to encode the current state and associated actions, as well as the set of possible transitions to other states [14].

As discussed in William Merryman's thesis [10], a given NFA can be converted into an equivalent DFA. The resulting DFA is not necessarily minimal. A DFA is said to be minimal if it has no redundant states. (i.e, if there is no other equivalent DFA with fewer states). This section presents a reduction algorithm that produces a minimal DFA recognizing the language  $L$  from any given DFA that recognizes  $L$ . The algorithm here generally follows the one presented by John E. Hopcroft and Jeffery D. Ullman [14].

In the remaining part of the presentation, we consider a DFA  $M$  to be a 5-tuple  $(Q, \Sigma, \delta, q_0, F)$ , where

- $Q$  is a finite set called the states,
- $\Sigma$  is a finite set called the alphabet,
- $\delta: Q \times \Sigma \rightarrow Q$  is the transition function,
- $q_0 \in Q$  is the start state, and

- $F \subseteq Q$  is the set of final states.

To accomplish the reduction, the notion of equivalent states and distinguishable states in a DFA are introduced.

### Equivalent States

Two states  $p$  and  $q$  are said to be equivalent if and only if for each input string  $x$ ,  $\delta(p, x)$  is an accepting state if and only if  $\delta(q, x)$  is an accepting state.

### Distinguishable States

For any two given states  $p$  and  $q$ ,  $p$  is said to be distinguishable from  $q$  if there exists some input string  $x$  such that  $\delta(p, x)$  belongs to the group of final states and  $\delta(q, x)$  does not, or vice versa.

### The Algorithm

The marking algorithm from [14] to minimize a given DFA is presented next. Without creating an entirely new DFA from scratch, this algorithm constructs the minimized version of the given DFA from the given DFA itself.

The marking algorithm is as follows:

**begin**

**for** every state  $p$  in the set of final states  $F$  and state  $q$  in the set of non final states  $Q - F$ , mark the pair  $(p, q)$ ;

**for** each pair of distinct states  $(p, q)$  in  $(F \times F)$  or  $((Q - F) \times (Q - F))$  do

**if** for some input symbol  $a$ ,  $(\delta(p, a), \delta(q, a))$  is marked then

**begin**

mark  $(p, q)$ ;

recursively mark all pairs on the list for  $(p, q)$  and on the list of other pairs that are marked at this step.

**end**

**else** /\*If no pair  $(\delta(p, a), \delta(q, a))$  is marked \*/

**for** all input symbols  $a$  do,

put  $(p, q)$  on the list for  $(\delta(p, a), \delta(q, a))$  unless

$\delta(p, a) = \delta(q, a)$

**end**

### Visualizing the algorithm

Since the above algorithm consists mostly of a collection of mathematical phrases, many students find it difficult to understand and interpret it in this form. The visualization software developed for this thesis is intended to help students understand the algorithm better. For the visualization process to be more helpful in this regard, every step in the algorithm is refined and filtered in such a way that it in fact represents a collection of steps.

#### Step 1 – removing useless states

The algorithm starts by first recognizing the useless states in the DFA and displays them in red. The algorithm used to recognize such states is the depth first search (DFS) algorithm presented in [15]. Although the book [14] goes through the marking algorithm first and then later removes the useless states, in this thesis we recognize and remove the useless states first, to save resources from being wasted. If there are any such states found by the algorithm, then the applet removes them. Else the applet moves to the next step of visualization.

#### Step 2 – identifying final and non final states

In step 3, the applet marks the pairs of states  $(p, q)$ , where state  $p$  belongs to the group of final states and state  $q$  belongs to the group of non final states. In order to visualize this step, before marking any of the pairs, the applet first breaks the states

of the DFA into two groups. One group consists of final states and the other consists of non-final states. The states belonging to the group of final states are represented in blue and those belonging to the group of non final states are left uncolored.

### Step 3 – initial marking of state pairs

In the next step the applet then clears out the blue color from the final states, thereby preparing the user for the next visualization. The applet then considers pairs of states one at a time such that every pair has one state from the group of final states and the other state from the group of non final states, and then marks such a pair.

Marking of states is one of the most important steps of the algorithm. This is done so that for any pair of states, at any given moment, it can be known whether they are candidates of being merged. In this step if a pair under consideration is marked, then they cannot be merged, since the marked pair will have one state from the group of final states and the other state from the group of non final states and are therefore distinguishable. And if they are not marked, then they are at this moment are candidates of being merged.

This step of marking is represented by showing the states within the pair under consideration as being colored in blue, if they should be marked. If the pair under consideration is not colored, then it indicates that that pair should not be marked. Once both the states in the pair being considered are processed, the applet clears this visualization making way for the next pair of states to be displayed.



#### Step 4 – determining distinct states

Once the marking of state pairs has been completed, the applet then advances to compute all distinct state pairs in the DFA. Since  $O(n^2)$  different colors would be needed to display the  $O(n^2)$  distinct state pairs for  $n$  states, the applet only highlights distinct pairs in succession with a single color (magenta) while simultaneously displaying these pairs in the text box.

#### Step 5 – completing the marking of states

For every pair of distinct states either belonging to the group of final states or to the group of non final states, it is checked to see if on any input symbol the pair transits to a pair of states that is marked. If it does, then the distinct states pair being considered is marked and all the pairs of states on the list associated with that distinct pair of states are marked. (and so on, recursively, according to the marking algorithm). If it does not, then that pair is put on the list of the pair of states to which it transits on the input being considered, unless both the states in that pair transit to the same state.

#### Step 6 – merging the redundant states

This step will separate the marked pairs of states from the state pairs that did not get marked. If a pair did not get marked, then the states within that pair will be merged. If a pair did get marked, then the states within that pair cannot be merged and will have to be present in the DFA as two individual states.

### Step 7 – reforming states and transitions

This step determines the transitions that will have to be created for the newly formed merged states, and arranges them. The transitions that were present for the original states that have been merged are removed in this step. Any other transitions that are present between the states of the newly merged state and the other states within the DFA, will be created to be consistent, but now pointing to and from the merged state instead of pointing to and from the states within that merged state.

### Step 8 – resetting the start state

Once all the transitions have been arranged, this step determines whether any of the merged states contain the start state. If so, this new merged state becomes the start state of the minimal DFA.

### Step 9 – removing individual states that were merged

The final step cleans up the DFA by removing all the individual states that are now part of the merged states. The resulting DFA will be the minimal one for the DFA started with.

In order to get the intuition behind this minimization process correct, the FSA applet of Grinder and Cogliati has been extended to incorporate visualizations of this algorithm. The next section provides a detailed description of this applet.

## DFA MINIMIZATION APPLET

The DFA minimization applet described in this section is based on Michael Grinder's and Josh Cogliati's FSA animator applets. The primary focus of this visualization was to be helpful to students falling under the category of being complete novices to that of being average at the concept.

The original applets were thus extended to include the *Minimize Demo* mode along with a *Directions* button that directs the user in using the applet. A brief explanation about the *Minimization Demo* mode and the *Directions* button is given below.

- The *Minimize Demo* mode allows the user to view the minimization process one step at a time, at a deliberate speed, and with every step being described in detail.
- The *Directions* button can be used before starting the minimization process to understand how to use the applet.

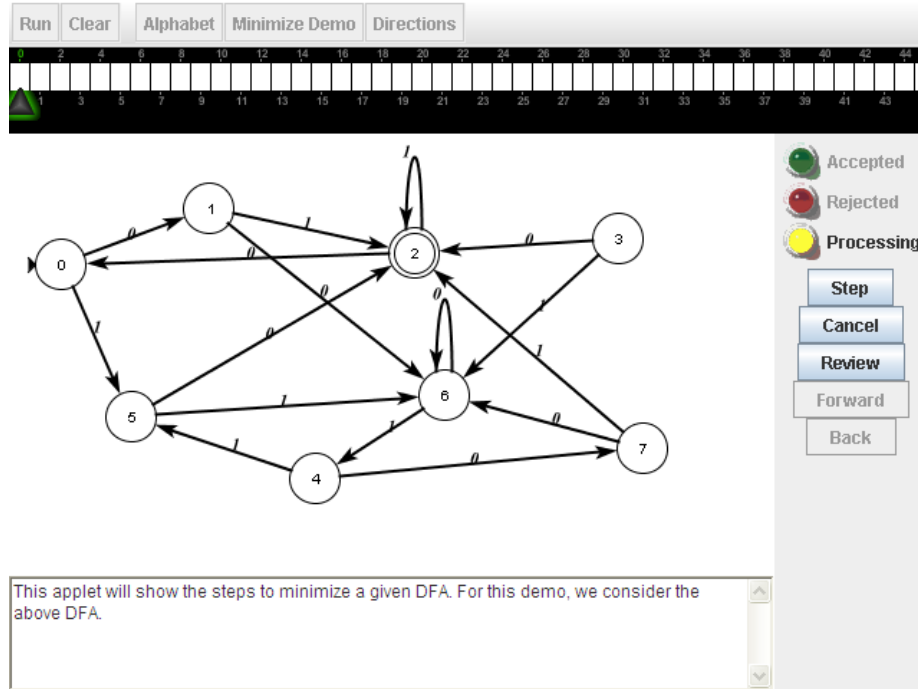


Figure 12. Start of Minimization demo.

### MINIMIZE DEMO VISUALIZATION

The visualization applet in this mode goes through the DFS algorithm to find and remove any useless states, and then follows the marking algorithm presented by John E. Hopcraft and Jeffery D. Ullman [14]. At every step, the visualization applet undergoes some visual changes, which are explained in the text box provided in the applet. The text at some steps also explains what changes are about to take place in the next step thereby preparing the reader to next look for those changes in the the displayed FSA.

Figure 12 displays the applet when the *Minimize Demo* button is first clicked. As seen in the figure, the action of clicking the *Minimize Demo* button deactivates (greys out) all the buttons at the top of the applet window corresponding to different modes in the applet. For example the buttons labelled *Run*, *Clear*, *Alphabet* and *Directions* which correspond to different modes in which the applet can be, are all deactivated. Once this is done, the applet starts executing the minimization algorithm one step at a time. In order to simplify the visualization process, the applet breaks down many individual steps in the algorithm into a set of simple steps.

Clicking on the *Step* button triggers the applet to move to the next visualization step in the algorithm. If there is a set of visualizations that are to occur in the next step, a description about these changes will be displayed within the text box of the applet during the current step.

Figure 13 displays the state of the applet when the *Step* button is clicked. This step displays the useless states in the DFA in red, and removes them from the DFA. If there are no such states, then the applet advances to the next step. Figure 13 displays a DFA with one such state displayed in red.

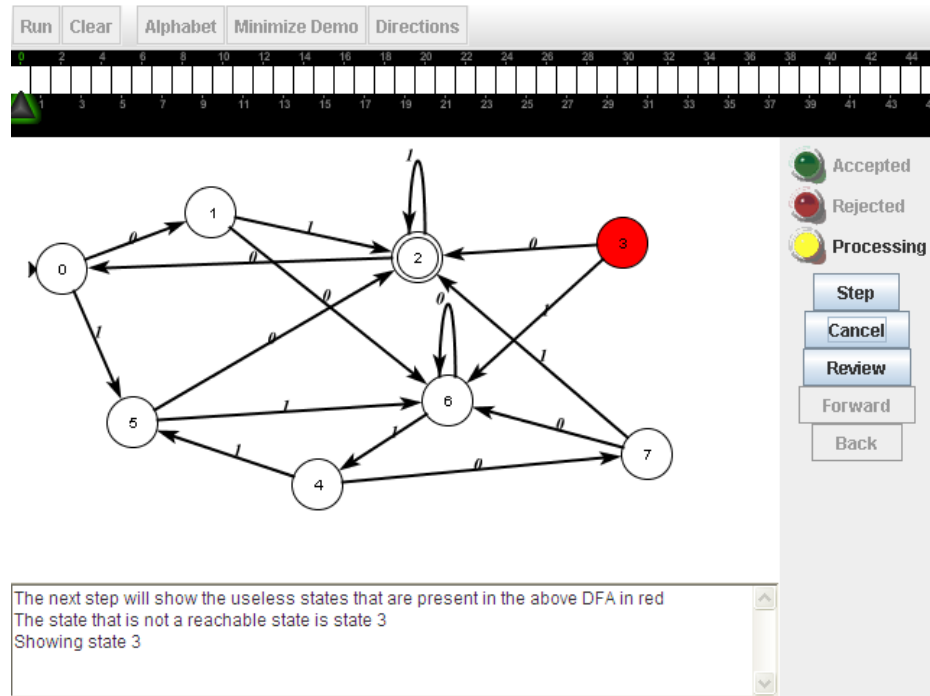


Figure 13. Applet showing useless states in red.

Clicking on the *Step* button will take the user to the next step in the algorithm. This step will display all the states that belong to the group of final states and the states that belong to the group of non final states. The applet visualizes this step by coloring the states belonging to the group of final states in blue.

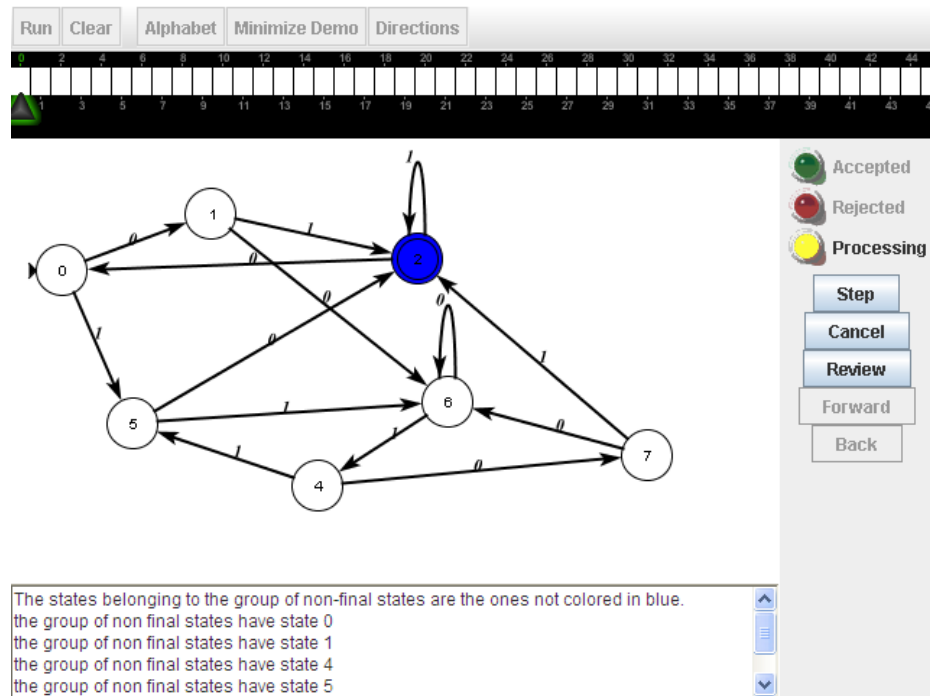


Figure 14. Applet showing final and non final states.

Once this is done, another click on the *Step* button causes the states that belong to the group of non final states (the ones that are not colored in blue) to be listed. The text box in figure 14 accordingly displays all the states that belong to the group of final states and non final states.

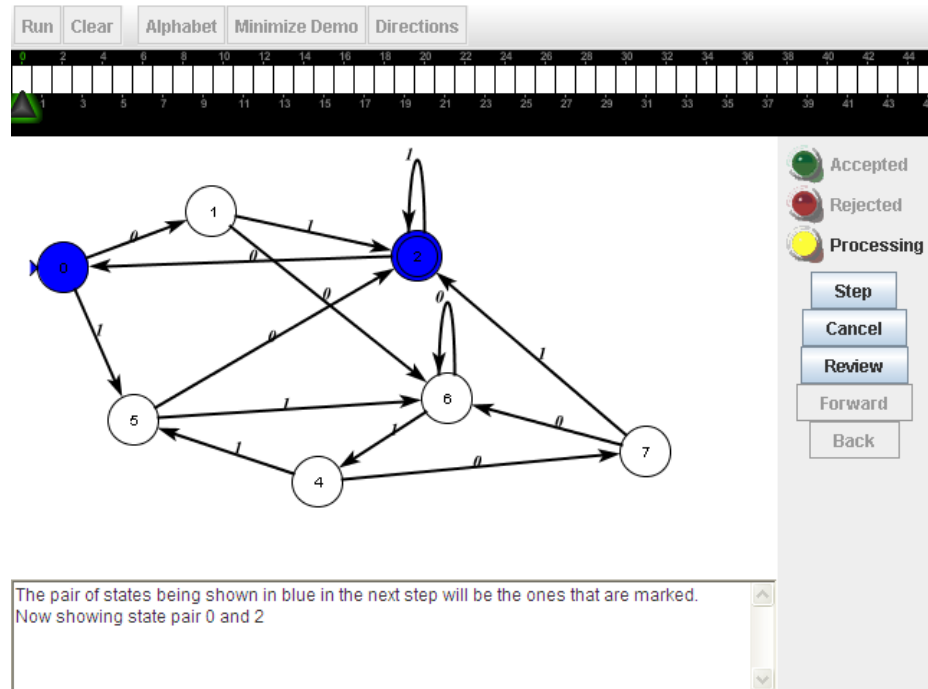


Figure 15. Applet showing the first pair of states being marked.

The applet now begins to mark all pairs of states such that the first state of a pair is a non final state and the second state is a final state. The states being considered for marking are both represented in blue at the same instant. Once a pair is highlighted in blue and listed in the text box, the applet then clears the highlighting and prepares the user to look for the next pair being considered for marking. Figure 15 displays the first pair of marked states.



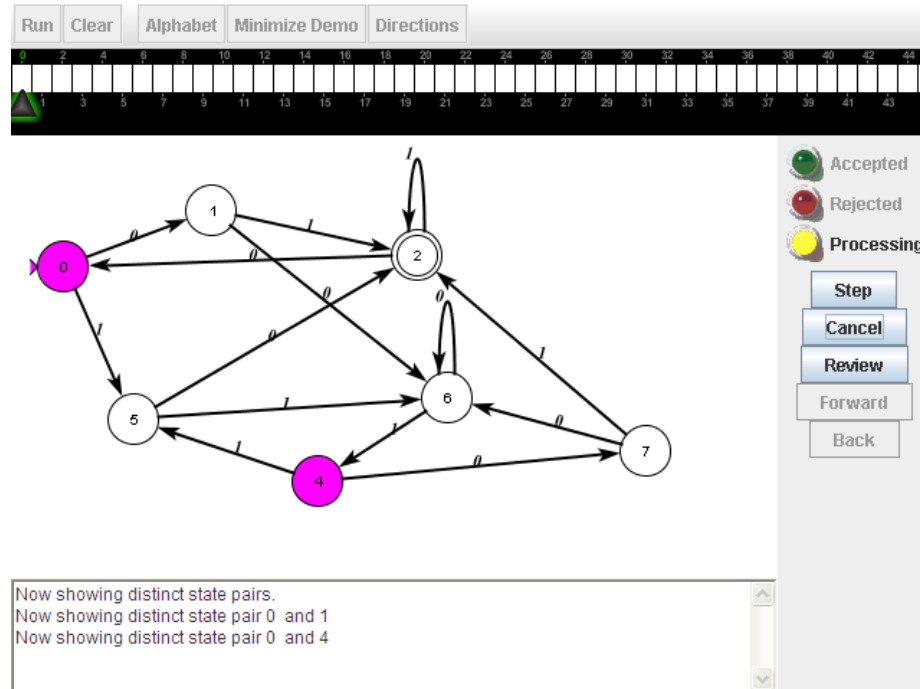


Figure 16. Applet showing pairs of distinct states.

Once all possible state pairs have been marked successfully, the highlighting is cleared. The algorithm then advances to the next step and finds all the distinct states from both the set of final states and the set of non final states within the automaton. The applet visualizes such distinct states in magenta. Figure 16 displays two such states. At this point, the algorithm finishes the task of marking distinguishable state pairs from the set of distinct state pairs identified in the previous step (see step 6 in the previous section “Visualizing the algorithm”).

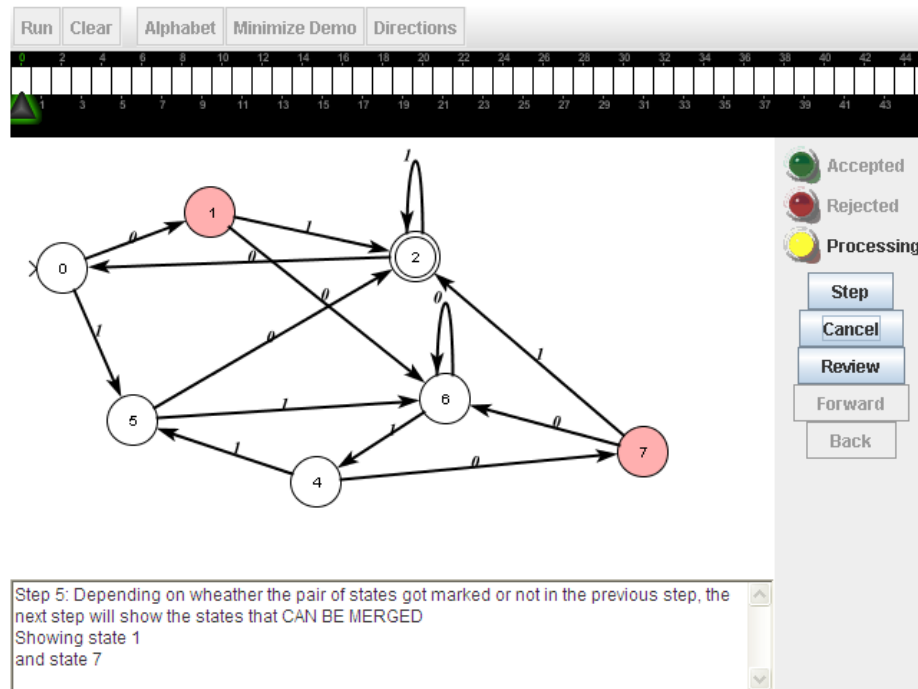


Figure 17. Applet displaying one of the pairs of states that can be merged.

After all distinguishable state pairs have been marked the next step checks to see if there are any pairs to be merged. These would be any state pairs not marked in the previous step. If there are, then the states that are to be merged are displayed in same color. Figure 17 displays the applet with one such pair in pink.

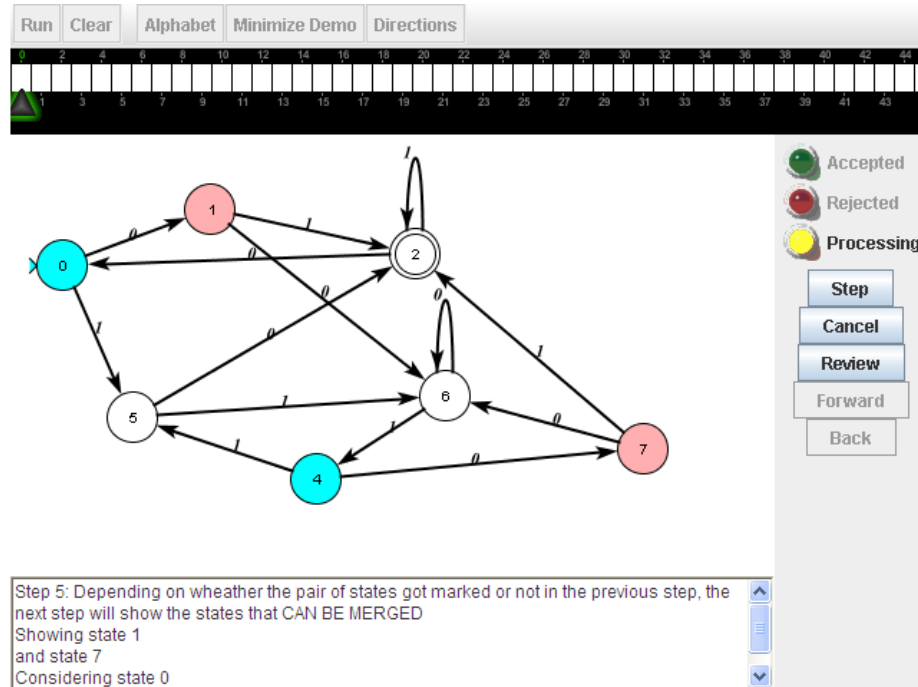


Figure 18. Applet displaying all pairs of states that can be merged.

Mergeable states are displayed with a click on the *Step* button. Once all the states that can be merged are displayed, a brief note about the states that are being merged is displayed within the text box. Figure 18 displays the state of the applet during this step.

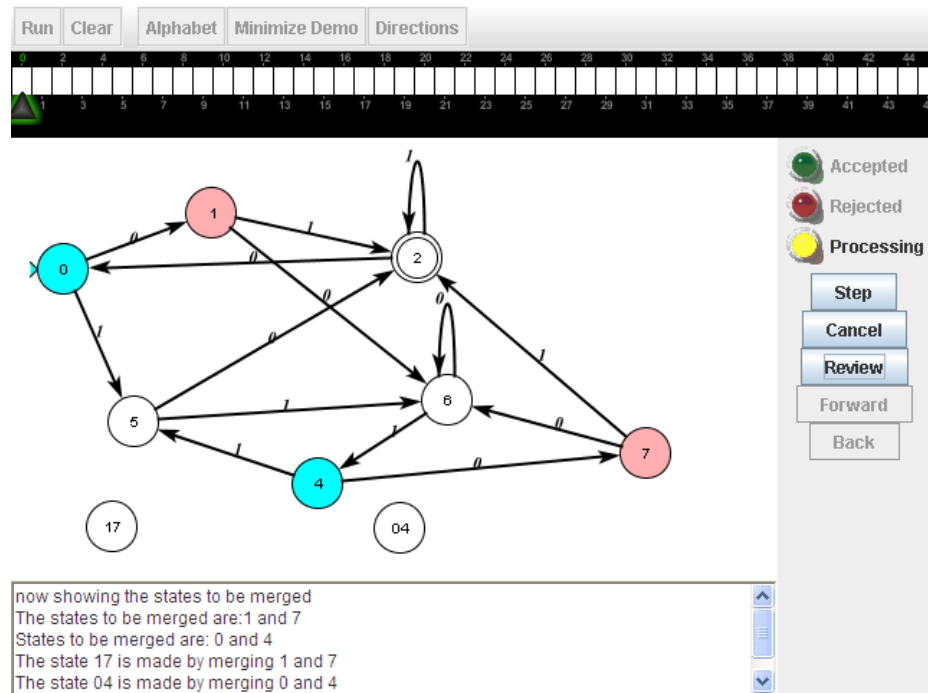


Figure 19. Applet displaying merged states.

The next step of the visualization creates new states that represent the merged states. The idea is to have one state represent states that are not distinguishable from each other as the same merged state. Accordingly, the new state will have a label that is a combination of the individual state labels. Figure 19 shows the applet displaying new merged states.

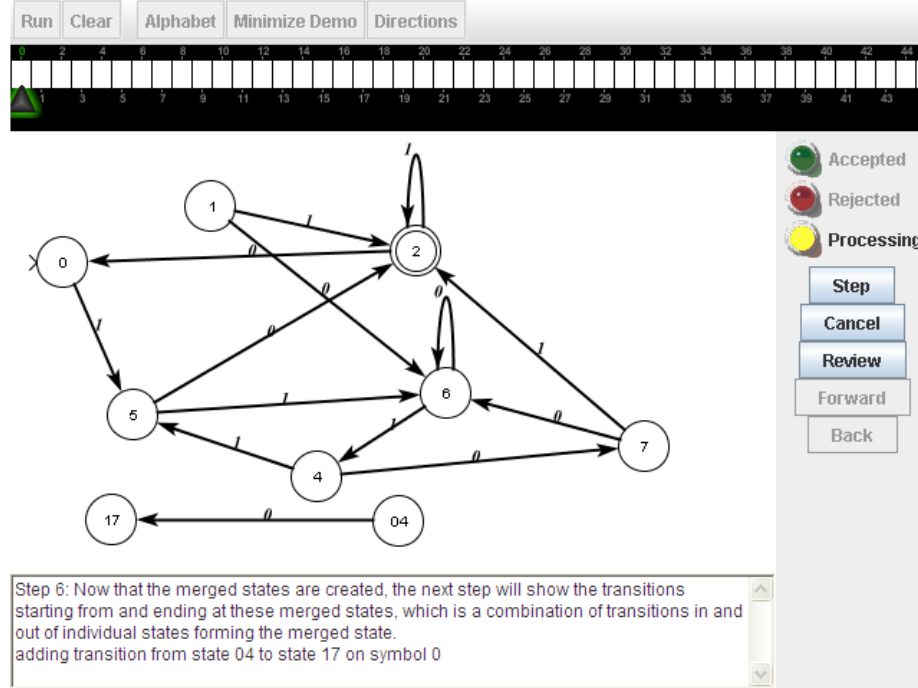


Figure 20. Applet displaying transitions within the merged states.

Once all merged states have been constructed and displayed, the next step will start computing and displaying the transitions into and out of these new merged states. The idea is to have the transitions that previously started from and ended at the individual states that now contribute to merged states, start and end at the new merged states. In order to visualize this step, transitions between the states that contribute to the merged states are displayed first. Since this visualization concentrates on the transitions among the merged states, the color in the states is removed. The original transitions are deleted from the applet and are applied to the merged states. Figure 20 shows the applet during this step.

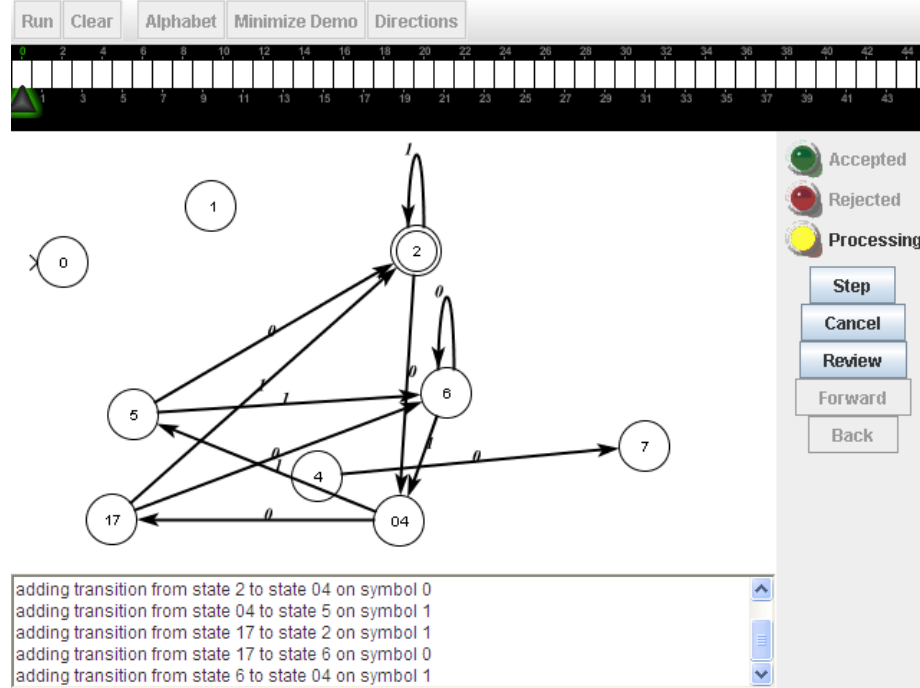


Figure 21. Applet displaying transitions among other states and the merged states.

As part of visualizing the step that creates transitions for the merged states, this step shows the transitions starting from states other than those in merged states but ending in merged states, and also those that start from merged states and end at other states. The applet will also display transitions that either start or end at only one of the states contributing the merged states. This is because the previous visualizations have handled transitions from and into the states contributing to the merged states but not the transitions that occur from states contributing to the merged state.

Figure 21 displays the applet showing the transitions at this step. From the figure, it can be seen that the DFA being considered has one such transition, it starts at state 4 and goes to state 7 although both 4 and 7 belong to the merged states

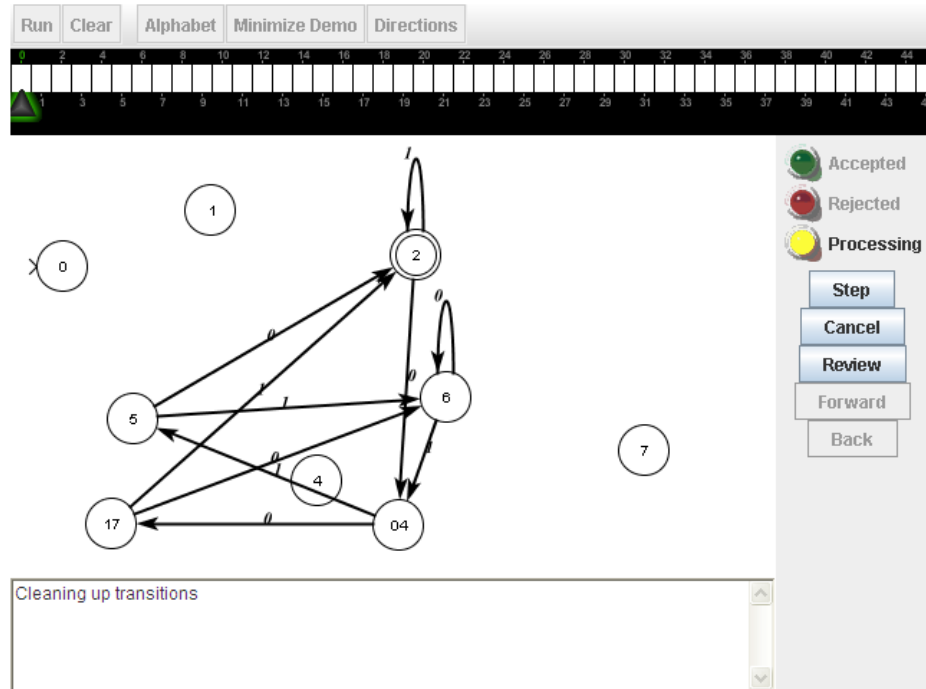


Figure 22. Applet cleaning up transitions among individual states that form merged states and other states.

labelled 04 and 17. A click on the *Step* button will remove such transitions from the DFA. Figure 22 displays the applet cleaning up such transitions.

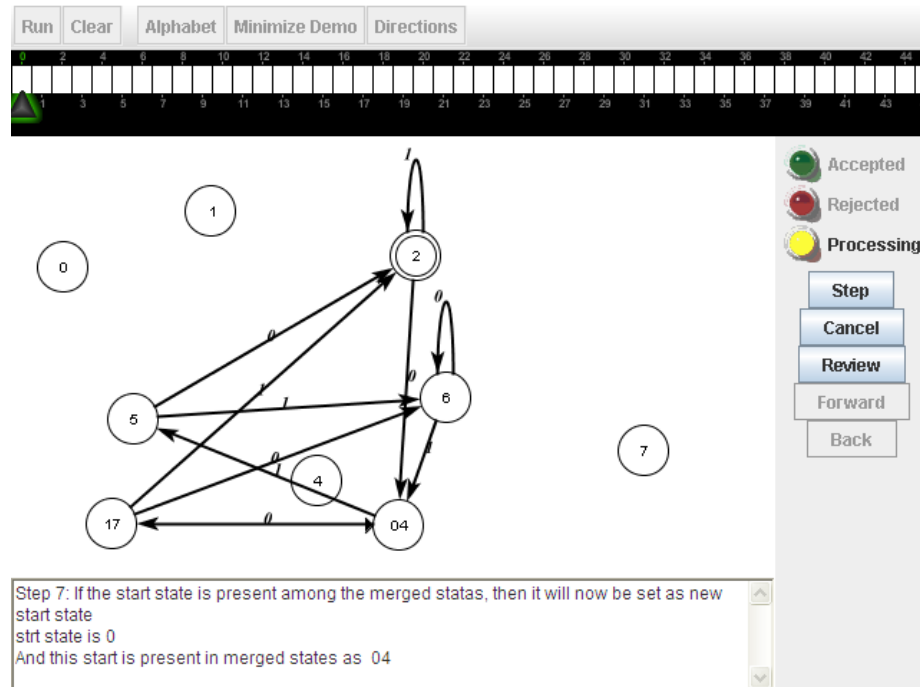


Figure 23. Applet displaying the new start state.

Once all the transitions have been recomputed, the next step checks to see if any of the merged states includes the start state of the DFA. If it does then the merged state including that start state is set as the new start state. If not, then the applet moves on to the next step which is to remove old individual states that are now included in new merged states. Figure 23 displays the applet with the new start state.



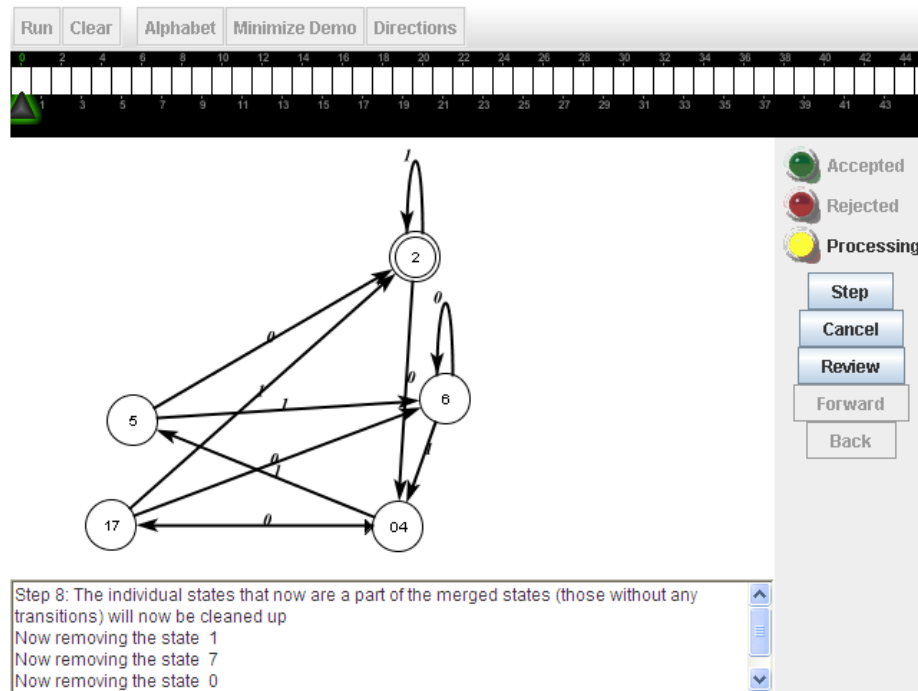


Figure 24. Applet with individual states which are now part of the merged states, being removed.

The next step is to delete all the individual states that are now part of the merged states; in other words, the ones that no longer have any transitions starting from or ending at them. Figure 24 displays the applet during this step.

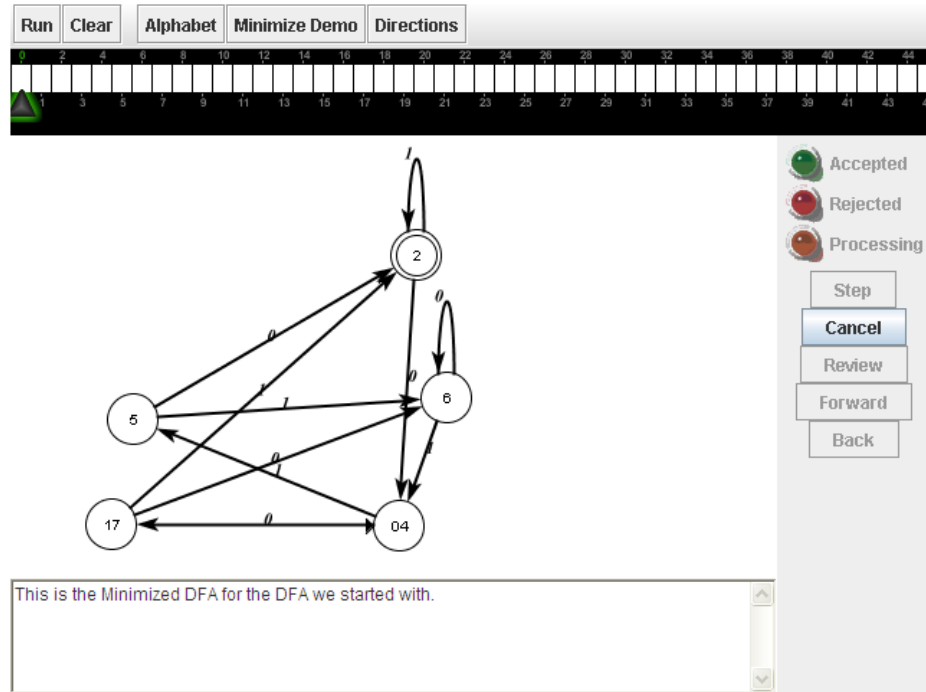


Figure 25. Applet displaying the minimized DFA.

Once all such states are deleted a click on the *Step* button reactivates the buttons on top of the applet and also deactivates the *Step* and the *Review* button. This confirms that the applet has completed the process of minimization and that the DFA presented on the applet at this moment is the minimized one. Figure 25 displays the applet at this step.

## CONCLUSIONS

As far as we are aware, the applet presented in this thesis is first to visualize the minimization of DFA's based on the algorithm presented in [14]. It is designed with visualizations that will help students to better understand the process of minimization according to the algorithm. The algorithm does not include some of the visualization steps presented in this thesis. The concept of useless states—states that do not lie on a path extending from the start state to a final state, is not part of the marking algorithm and was implemented using a DFS algorithm from [15].

The applet accepts as its input any DFA in order for the minimization process to begin. This allows the students the flexibility of having any DFA that they feel will make understanding the minimization process easier, as input to the applet. The applet will only work if the input automaton is a completely specified DFA.

Also, the text box at the bottom of the applet provides the user with detailed descriptions about what is happening at the current step and what is to be expected in the next step. It also explains how the current step is being visualized and what that particular visualization includes.

## EVALUATION

Student evaluations of the use of visualization applets are needed to determine their effectiveness. This thesis was unfortunately completed during a semester when the theory of computing course was not offered. However as part of the background work for this thesis, the author did help in the evaluation of a related applet at the time when the course was taught earlier.

### The Regular Expression Animator

The applet used for the evaluation process was the regular expression applet of Walsh [9] and Pascoe [1]. A control group and a test group were established for the course. The control group had a traditional textbook and lectures with which to learn the concept. A set of questions, called the pre-evaluation questions, was given to the group as an assignment that tested the level at which the students understood the concept. The students could use their textbook and class notes and were given the entire class hour to complete the assignment.

The test group received similar treatment (i.e, they were given the same pre-evaluation questions). Afterwards, they were allowed to use the regular expression animation applet. Once comfortable with the applet, the students were then given with the same assignment as the control group and were given the same amount of time to complete it.

The results of these questions were evaluated for both the groups and a comparison about the performance of every student as part of the control group and the test group was made. Depending on the highest score received by a student as part of a group, it was determined whether the animation applet was helpful for that student or not.

The average score of all the students in each group was also computed. The test group scored the highest. No attempt was made at that time to determine if the difference was significant.

### Evaluating the DFA Minimization Applet

The theory of computation course is taught once a year at MSU and since, as noted, this thesis was completed at the time when the course was not being taught, evaluating the performance of the minimization applet is yet to be done. However, a set of questions that could be used for pre and post testing were developed for use with test and control group students (see appendix A). The intent is to present the pre-test to both the groups just after the process of minimization is introduced to them. Control group and test group students would both have access to the textbook and lectures.

After the pre-test, both the groups would then be given the same assignment to complete. The control group would be able to use the class textbook and lecture notes. The test group would be able to use the class textbook, lecture notes, and the DFA minimization applet. An example assignment can be found in Appendix

B. After the assignment were complete, both groups would be given the post-test to identify any differences in student learning success and attitudes.

As part of the projected evaluation process, a questionnaire was also developed to identify any issues with applet use. Appendix C provides a sample of this feedback questionnaire. This questionnaire would be presented to all the students after the assignment had been completed and when every one would have had the chance to experiment with the applet.

## FUTURE DIRECTIONS

Two major and one minor extensions of the applet would increase its usefulness.

### Exercise Mode

As it now stands, the DFA minimization applet involves only passive learning. Extending it to include active learning by engaging the students in the minimization process will improve its usefulness. The idea is to engage the students in the minimization process by having them select distinguishable states, for example, and then providing them with immediate feedback on their answers. The second major thing to be done is to include the construction of the distinguishable states table (see [14]) along with the steps currently displayed. This extension requires a major overhauling of the FSA applet and is left for another to do.

### Minor Extensions

There are also other minor things that could be looked at: the refreshing of the minimization applet when the *Refresh* option on the browser is clicked, and the working of the *Review* button. The minimization applet does refresh in the foreground but does not clean up the data structures associated with the applet animation, which causes unexpected results to show up.

Presently when the *Review* button is selected to review any step in the minimization process, the applet does not function in the way it is supposed to. This is due

to the way the applet was threaded that causes resource conflicts. Once this issue is fixed all those animated applets that do not include this feature can add it by adding a little bit of code, hence making it possible for the user to review any step at any point in the animation.



## BIBLIOGRAPHY

- [1] Joshua J. Cogliati, Frances W. Goosey, Michael T. Grinder, Bradley A. Pascoe, and Rockford J. Ross, *Realizing the Promise of Visualization in the Theory of Computing*, Journal of Educational Resources in computing, July 2004.
- [2] Rockford J. Ross, *Hypertextbooks: Animated, active learning, comprehensive teaching and learning resources for the web*. Software Visualization, pages 269-283, 2002.
- [3] Colleen Kehoe, John Stasko and Ashley Taylor, *Rethinking the evaluation of algorithm animations as learning aids: an observational study*. *Int. J. Human-Computer Studies* (2001) 54:265-284. <http://www.idealibrary.com>
- [4] Kenneth James Williams Craik, *The Nature of Explanation*, Cambridge Univ. Press, Cambridge, 1943.
- [5] Phil Johnson Laird and Ruth Byrne, *Mental Models- A Gentle Introduction*, available on WWW at [http://www.tcd.ie/Psychology/Ruth\\_Byrne/mental\\_models](http://www.tcd.ie/Psychology/Ruth_Byrne/mental_models).
- [6] Albert Bandura, *Social foundations of thought and action: A social cognitive theory*, Englewood Cliffs, NJ: Prentice Hall, 1986.
- [7] Michael Thomas Grinder, *Active Learning Animations for the Theory of Computation* PhD thesis, Montana State University, Computer Science Department, December 2002.
- [8] Michael Thomas Grinder, *A preliminary empirical evaluation of the effectiveness of a Finite state automaton animator*, Twenty fourth SIGCSE Technical Symposium on Computer Science Education, 35:157-161, March 2003.
- [9] Michael Thomas Grinder, S. B. Kim, Teresa L. Lutey, Rockford J. Ross, and Katie F. Walsh *Loving to learn theory: Active learning modules for the theory of computing*, Thirty third SIGCSE Technical Symposium on Computer Science Education, 34(1):371-375, February 2002.
- [10] William Patrick Merryman. *Animating the Conversion of Nondeterministic Finite State Automata to Deterministic Finite State Automata*, December 2006.
- [11] Susan Rogers. *Java Formal Language Automata Package (JFLAP)*, February 2006. <http://www.jflap.org>

- [12] Susan Rogers and Thomas W. Finley. *JFLAP: An Interactive Formal Languages and Automata Package*. Jones and Bartlett Publishers, 2006.
- [13] Peter Linz. *An Introduction to Automata and Formal Languages, 4ed.* Jones and Bartlett Publishers, 2006.
- [14] John E. Hopcroft, Jeffery D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison- Wesley Publishing Company, 1979.
- [15] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts London, England. McGraw- Hill Book Company New York St. Louis San Francisco Montreal Toronto, 1990.

## APPENDICES

APPENDIX A

SAMPLE PRE AND POST TEST

SAMPLE QUESTIONS FOR PRE-AND POST-TEST

1. Briefly describe what is meant by a “Minimal DFA”.
2. Is every DFA minimal? Explain your answer.
3. Can every DFA be minimized? Explain why.
4. What do you understand by the term “equivalent” states?
5. What do you understand by the term “distinguishable” states?
6. What do you understand by the term “merged states”?

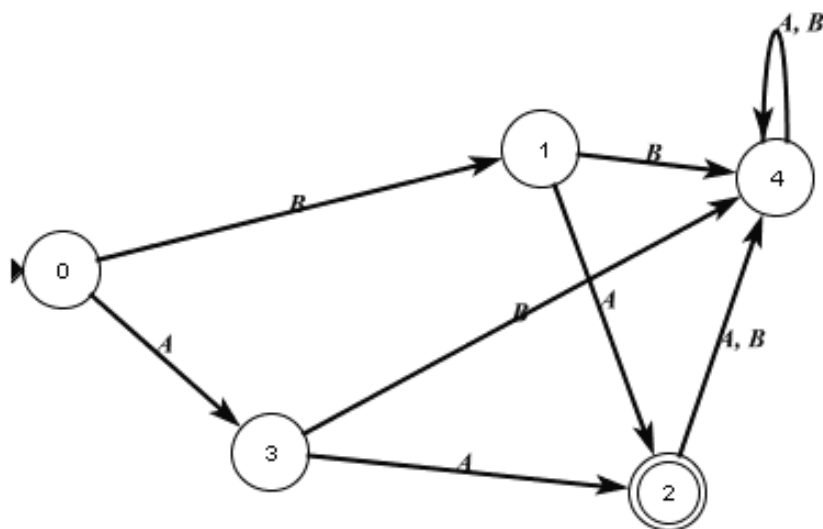


Figure 26. Sample DFA.

7. Find all the distinguishable states in figure 26, and explain why they are distinguishable.
8. Are there any states in the DFA presented in figure 26, that are mergeable?  
Explain your answer.
9. Draw the equivalent minimal DFA for the DFA given in figure 26.

APPENDIX B

ASSIGNMENT SAMPLE

SAMPLE ASSIGNMENT QUESTIONS

Assume that a DFA is given;

1. List a state, if present, that can be termed “useless”.
2. Identify the distinguishable states and list them.
3. Describe in your own words,

The term “marking” pairs of states

The importance of marking pairs of states

The importance of associating a pair of states with a list.

4. List the states that can be candidates of being merged and give reasons for selecting them.



APPENDIX C

FORMATIVE FEEDBACK QUESTIONNAIRE

SAMPLE FORMATIVEFEEDBACK QUESTIONNAIRE

1. Did the applet make learning more time efficient than a hardcopy textbook?

- (a) Strongly Agree
- (b) Agree
- (c) Negligible
- (d) Disagree
- (e) Strongly Disagree

2. Was learning the minimization process through the applet more enjoyable than learning from a traditional textbook?

- (a) Strongly Agree
- (b) Agree
- (c) Negligible
- (d) Disagree
- (e) Strongly Disagree

3. Was the applet loading time really long and frustrating?

- (a) Strongly Agree
- (b) Agree
- (c) Negligible
- (d) Disagree
- (e) Strongly Disagree

4. Were the controls on the applet all functional

- (a) Yes
- (b) No, list the non functional items

5. Overall, was your experience using the applet positive?

- (a) Strongly Agree
- (b) Agree
- (c) Negligible
- (d) Disagree
- (e) Strongly Disagree