jFAST: A Java Finite Automata Simulator

Timothy M. White and Thomas P. Way
Applied Computing Technology Laboratory
Department of Computing Sciences
Villanova University
Villanova, PA 19085

timothy.m.white@villanova.edu
thomas.way@villanova.edu

ABSTRACT
Visualization and interactivity are valuable active learning techniques that can improve mastery of difficult concepts. In this paper we describe jFAST, an easy-to-use graphical software tool for teachers and students, with an emphasis on introductory level finite state machine topics. The jFAST software enables visual design, exploration and simulation of a variety of finite state machines, with a primary goal of enhancing teaching effectiveness in this subject, particularly for less advanced computer science students. The architecture and functionality of jFAST are explained, and results of preliminary evaluation are provided.

Categories and Subject Descriptors
F.1.1 [**Theory of Computation**]: COMPUTATION BY ABSTRACT DEVICES – *Models of Computation* D.1.7 [**Software**]: PROGRAMMING TECHNIQUES – *Visual Programming.*

General Terms
Theory, Design, Languages.

Keywords
Finite automata, simulation, educational software, finite state machines, theory of computation education.

1. INTRODUCTION
Within a computer science curriculum, courses in theory are often perceived by students as the most difficult to understand and do well in. Some survive the experience with only minimal understanding of the formal languages and automata (FLA) material that is vital to subsequent courses in programming languages and compilers. Computer science students generally enjoy writing computer programs, a reality that can be used to improve learning in FLA courses. This dovetails nicely with accepted ideas of active learning, where the results of static activities such as lectures and pencil-and-paper assignments can be greatly enhanced with hands-on activities [3]. Active learning is particularly important in computer science, enabling students to master difficult conceptual material in many subject areas [1].

SIGCSE'06, March 1–5, 2006, Houston, Texas, USA.

The study of Finite State Machines (FSMs) is pervasive in FLA courses, as well as those in engineering and mathematics, and is also a concept that is very difficult for students to fully visualize on a whiteboard or notebook. Recognizing this, significant effort in recent years has gone into developing educational software that provides a more interactive FSM learning experience. There are a number of FSM simulators available on the internet [2,4,6,7,9], although most suffer from one or more flaws that make them less than ideal as learning tools, particularly for less advanced students. Some are quite powerful, but do not provide a convenient mechanism for displaying and visually simulating the FSMs as would be ideal in a classroom setting [8]. Some are difficult to setup and use [7] or lack visual clarity or dynamic simulation capability [2]. Additionally, almost all have been designed as tools for advanced students, under the assumption that the students have already grasped the fundamental concepts, and are dependent on advanced mathematical and idiosyncratic user interactions [4,9]. Finally, because many assume users are advanced they rely on the arcane, if appropriate, symbols of FSMs, making the creation of complex FSMs a difficult and often confusing task for less experienced users [6,7].

This paper describes the design and utility of the Java Finite Automata Simulation Tool (jFAST), an instructional software package intended as an easy-to-use, easy-to-learn software tool for teachers and students for discovery and exploration of finite state machines. It is designed to be a complementary alternative to the more advanced and widely used JFLAP tool developed at Duke University [5,9]. The jFAST software enables teachers to create simple or complex FSMs that could then be displayed to students via a classroom projector. jFAST also allows teachers to distribute FSMs to students via email, with students then able to actively learn through hands-on manipulation and visual simulation. Students can create their own FSMs using the familiar and intuitive drag-and-drop graphical user interface conventions of modern computing.

To support educational use of jFAST, a web site has been designed to distribute the software and a wide variety of example FSMs. Over time, the web site will collect jFAST-compatible FSMs that implement examples from the most popular and widely-used FLA course textbooks, subject to permission of the authors.

2. jFAST ARCHITECTURE

jFAST is a robust, intuitive and portable application that allows the user to quickly and easily design, modify, and graphically animate finite automata. Functionally, it allows a user to create a graphical automaton from scratch, giving the designer the power to easily create complex automata much like creating a drawing in a modern graphics program, such as free tools like XFig and OpenOffice Draw, or commercial products like Adobe Illustrator or CorelDraw. These tools provide excellent graphics capability for drawing automata, but do not support animation.

Creating a new automaton in jFAST is straightforward. The user first clicks with the mouse to add one or more states. Additional properties of a particular state can be specified exactly as one would access the properties of any Windows object, by right-clicking on the desired element to open a pop-up dialog box. This Windows-interface convention is followed for all basic components of the automaton. Users then add, modify, and remove an alphabet, transitions, and labels to or from the basic automaton diagram. jFAST utilizes the object-oriented design and principles of Java to derive a conceptual FSM from the design, which is then used for simulating and testing the FSM on different inputs as specified by the user.

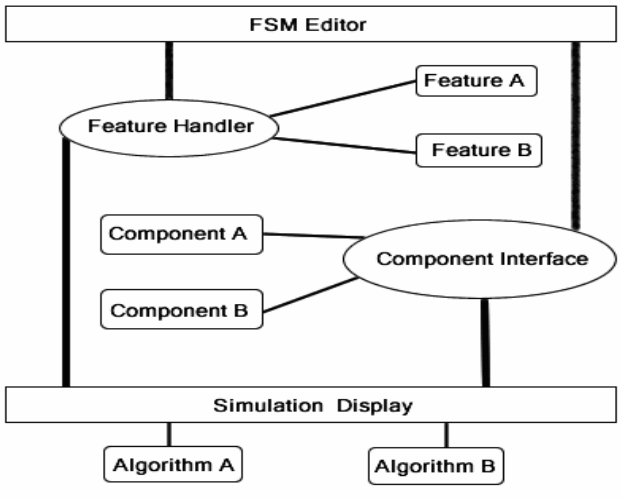

Figure 1. Diagram of jFAST architecture.

The architecture of jFAST is illustrated in Figure 1. The FSM Editor is a user interface that controls creation, modification and display of FSMs as the user designs them. The Feature Handler and Component Interface provide a high-level interface to the Features (user interactions with and manipulations of FSMs) and Components, coordinating the interactions among the various on-screen elements of each FSM. The Simulation Display is also a user interface that uses the Feature Handler and Component Interface in a read-only mode for graphically displaying an FSM in motion as they simulate one of the Algorithms.

jFAST is implemented purely in Java, using Swing for all user interface components, to make it as widely portable as possible. Each component of an FSM (State, Transition, Label, Alphabet) is implemented as a customized JComponent, and therefore is responsible for drawing itself on the screen. Before running the FSM on an input, jFAST compiles the object-oriented representation into an array-driven logical form that is then simulated on any given input of arbitrary length. This also allows jFAST to quickly and accurately generate a non-deterministic search tree representing all possible paths that the automaton would take over a given input, making it adept at handling even extensive non-deterministic FSMs. In order to perform the simulation, a list of possible simulation paths is first generated. These paths then are used as templates for efficient graphical simulation or interactive exploration of a given FSM.

Integral to jFAST, and bundled with the software distribution, is a suite of finite state machines of various types (i.e., DFA, NFA, Pushdown Automata, State Machines, and Turing machines) so that teachers and students may use the same program for an entire course. All supported machines are available to the user from the same interface, a feature not supported in some similar programs [2,4,6,7]. In addition to this multi-machine compatibility, jFAST is designed for use in a classroom environment, with clean and simple graphics and the ability to pause simulations and step through transitions one at a time. Labels are easy to add and modify to clarify the presentation of the machine. The speed and flexibility of design and interactive simulation controls in jFAST provide a time-saving alternative to custom-designed PowerPoint animations that can require hours of tedious design effort.

To support independent learning, jFAST has a design error checking feature. When invoked, the error checker analyzes the current FSM and provides descriptive messages about any mistakes found with suggestions on how to correct them. The error checker doubles as a grading tool, providing teachers with an easy way to automatically analyze an FSM for mistakes.

The goal of the jFAST architecture is to provide an easy platform for learning about FSM through interactive design and graphical simulation. For introductory level students, the familiar user interface and drag-and-drop approach to design minimizes the learning curve. Students can quickly get to the task of designing and simulating FSMs. For more advanced students, jFAST supports arbitrarily complex FSMs of many types. Instructors can assign homework or programming projects that rely on jFAST, collect the results via email, and assess them using jFAST.

2.1 Functional Design Elements

The features incorporated in jFAST are described here to illustrate the design considerations and issues involved in constructing this form of educational software, in this case to support usability in and applicability to an FLA course:

Drag-and-drop creation of FSMs - Creating an FSM is accomplished by using a mouse to drag states to desired positions on the screen, clicking and drawing transitions that connect states, and right-clicking on any element to modify its properties. All objects are selectable and editable throughout creation process; objects can be created and modified in any order.

Easy display and simulation of FSMs - Through a separate simulation screen, an FSM is displayed and simulated. jFAST can run an FSM using preloaded input, or input can be provided interactively. Intuitive controls are provided for starting, stopping and stepping through the simulation.

Integration of sub-automata - jFAST provides a modular approach to creation of complex automata through sub-automata. An FSM can be created in a number of parts, or sub-automata, each of which is a distinct FSM. These sub-automata can then be merged to form more complex FSMs.

Saving and exporting of FSMs – The FSMs created in jFAST can be saved in an XML file format to be shared with others or reloaded later for continued modification and simulation. In addition, FSMs can be exported as JPEG image files for use in documents or for display.

Zoom in and out - FSMs can be displayed with smaller or larger components, effectively allowing the viewpoint of the user to zoom in or out. These automata can be as made as large as desired, with automatic scrolling enabled when needed.

Multiple types of FSMs supported – DFA, NFA, Pushdown Automata, State machines, and Turing machines are all supported, allowing for more complicated types of FSMs to be created and explored with the same ease of use and rich features as deterministic finite automata.

User help and tutorials – Complete application help and tutorials are provided to users via a menu selection. This helpful information includes instructions on how to use jFAST and explanations of the different theories and FSMs that are implemented in the application.

2.2 Design and Simulation Interfaces

jFAST supports an intuitive graphical drag and drop design editor, and equally easy-to-use simulation environment.

Design Interface

The design interface is shown in Figure 2. States and labels are displayed on a simple canvas at a user chosen location, and can be relocated at will. When states are relocated, the transitions connecting states automatically redraw themselves to maintain their clarity and visibility. All components highlight themselves when they are the focus of the mouse, to enable easier navigation of complex FSMs. The design editor provides a toolbar to modify the open automaton along with a supplementary menu. All menu items have simple and intuitive keyboard shortcuts.

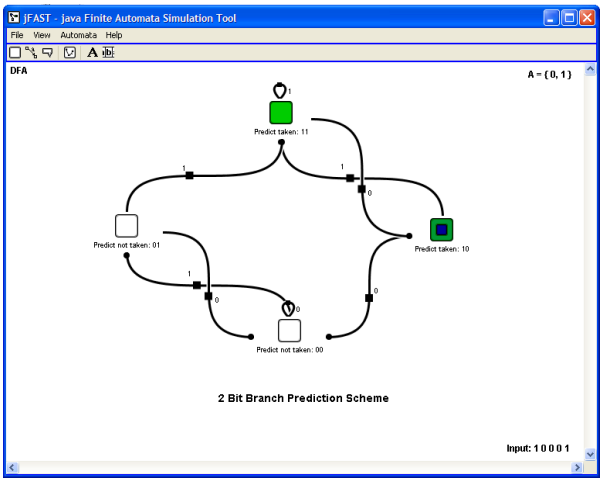

Figure 2. jFAST design interface.

The FSM design and editing interface has the following three main components:

1. The **menu bar**, which provides access to less frequently used functionality, such as adjusting the display settings, loading and saving an FSM, invoking the error checker, and reading the help information and tutorials;
2. The **tool bar**, which provides icon buttons for FSM modification tools. The available tools are: state tool, transition tool, label tool, sub-automaton loading tool, alphabet modification tool, and input modification tool.
3. The **automaton display** is a large white canvas, used for laying out an FSM, it also displays additional information about the current automaton. The type of FSM is shown in the upper left corner of the display, the current alphabet in the upper right corner, and the current input in the lower right corner.

Simulation Interface

The simulation interface resembles the automaton display of the design interface, and is used to display the continuous or step-wise simulation of an FSM for a specified input. The simulation environment optimizes the available display space, centering an FSM on the screen. The relocatable toolbar displays the current input and other simulation data, and provides controls for exploration of an automaton interactively. An example FSM displayed on the simulation interface is shown in Figure 3.

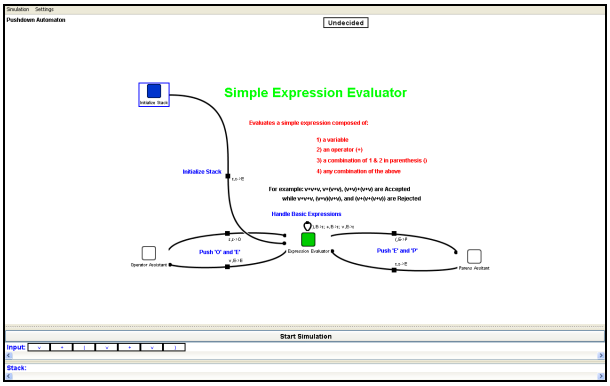

Figure 3. jFAST simulation interface.

The simulation interface consists of three components:

1. The **menu bar** provides access to basic configuration and display settings; the input/interaction bar; and return to the automaton display.
2. The **input/interaction bar** at the bottom displays current simulation input for the automatic mode, or buttons for the alphabet symbols for the interactive, stepwise simulation mode. The input/interaction bar also displays other extant information for the automaton, such as the state of the stack or Turing machine tape.
3. The **simulation display** handles display of the FSMs, and does not allow modification of the automaton. Layout is adjusted for clear viewing.

2.3 Machine Simulations

jFAST supports the simulation of a variety of common finite state machines. The FSMs implemented in jFAST consist of: DFA, NFA, Pushdown Automata, Turing Machine and State Machine. A brief description and a screen capture is provided below to illustrate each within the context of the jFAST interfaces.

DFA

Deterministic Finite Automata are a basic type of FSM in which each state has exactly one transition for every symbol on the alphabet (see Figure 2). jFAST enforces all deterministic aspects of DFA and dynamically detects non-deterministic aspects, providing helpful feedback for correcting basic errors.

NFA

Non-deterministic Finite Automata are a basic type of FSM in which non-determinism is allowed (Figure 4). jFAST handles numerous forms of non-deterministic behavior such as multiple transitions for a symbol, no transitions for a symbol, and ε-transitions (i.e., "epsilon edges," transitions taken without consuming any input).

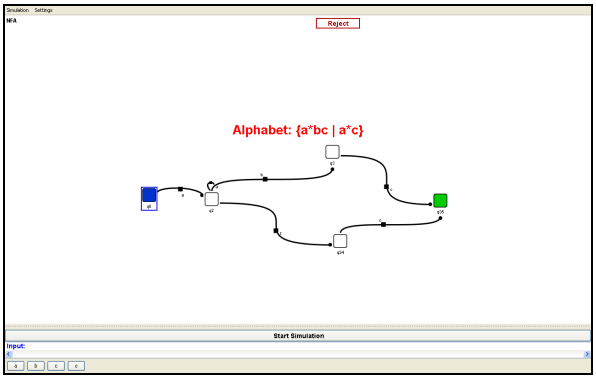

Figure 4. Example NFA shown in simulation interface.

Pushdown Automata

Pushdown Finite Automata are basic non-deterministic FSMs which have access to a simple stack (Figure 5). jFAST assumes acceptance is predicated on an empty stack, which is equivalent to other acceptance methods, achieved using of ε-transitions.

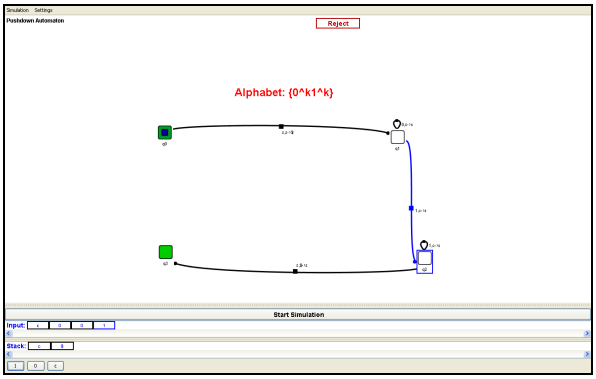

Figure 5. Example of a Pushdown Automaton.

Turing Machine

The Turing machine is a complex model of computation in which a non-deterministic automaton has access to a tape of infinite capacity (Figure 6). jFAST supports serial tape access in either direction, as well as providing the 'stay' option, and can easily be extended to support multiple tapes.

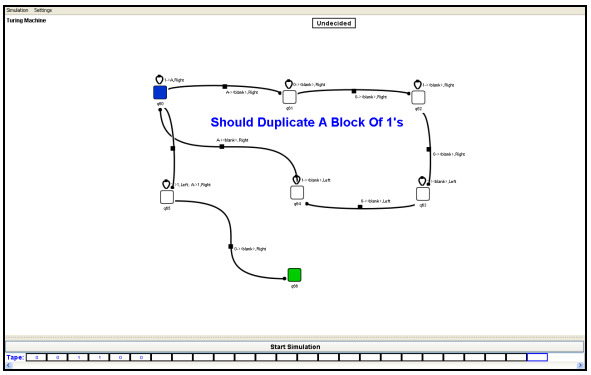

Figure 6. Example of Turing Machine.

State Machine

A State machine is a simple model of computation that represents the abstract state of a machine (Figure 7). jFAST allows for easy creation of, and interaction with, arbitrarily complex State Machines by providing buttons at each state representing each possible exit transition. jFAST automatically derives a 'state machine alphabet' from the transitions over the machine, and uses this to allow in-depth exploration.

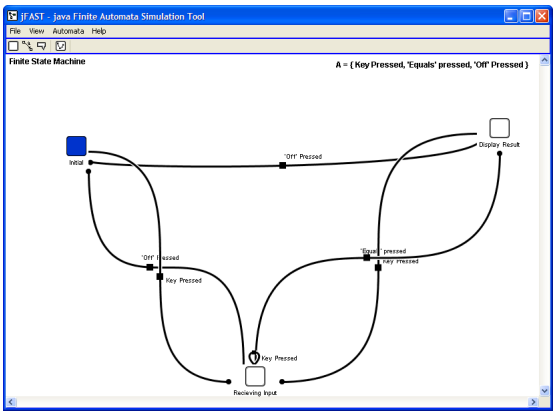

Figure 7. Example of State Machine.

2.4 Extensibility and Support

The jFAST software architecture was designed to be easy to maintain and customize. All interactions between the components of an FSM and the simulation and presentation software is guided by defined object-oriented Java interface classes, making the addition of new components as simple as implementing a few methods and reusing (or re-implementing) the painting functionality. The simulation algorithms are implemented as classes that extend a fundamental Simulator class.

The simulator for a new type of machine requires overriding the actual simulation mechanics, only; a new simulation algorithm can be added without any modification to other parts of the software. Thus, adding an additional type of FSM to jFAST involves editing the Java code to add a new menu choice and implementing standardized interfaces to the machine's components, including the simulation logic. Because the design is object-oriented, new features are added by implementing the new feature as a new Java class, and then integrated by adding a small segment of code to the interaction manager.

3. EVALUATION

jFAST was developed as a tool for teachers and students, and has been evaluated by each in a pilot study to assess the usability and potential effectiveness of the software. Three college professors and one high school teacher, all experienced in teaching material covered in FLA courses, evaluated jFAST. All were enthusiastic about its potential benefit to their students. The reviewers found the software easy to use, and were quickly able to design and simulate examples of FSMs used in their classes. The drag-and-drop approach to FSM creation was key to its perceived value to enabling students to focus on concepts rather than the mechanics of a new computer program. As one professor responded, "There are various simulators available, of course, but none fully convenient enough to be worth the trouble - the time overhead for students to figure out all annoying technical details at the expense of studying what is theory in the proper sense. So, simplicity and convenience of such tools may be crucial for a Theory instructor in deciding whether to start using them in his or her class. And simplicity and convenience is exactly what I like about jFAST."

Several teachers expressed the desire for more types of FSMs to be supported, such as Turing machines that accept or reject input rather than solely providing output, or having more control over selecting the algorithms used in simulating the various FSMs. Furthermore, all the professors requested a more extensive web site detailing the use and utility of jFAST, remarking that having supporting information readily available was as important as the design and reliability of the software.

The study included a more formal user evaluation in which 18 college students enrolled in a junior-level computer science course were asked to explore the jFAST software after a brief introduction. Students then completed a questionnaire describing their prior experience with FLA, their experiences in Computer Theory classes, and their reactions to jFAST. Many students (77%) were unsatisfied with their understanding of FLA and their experiences in formal Theory, with unanimous agreement that using a tool like jFAST would be a benefit. jFAST addressed student concerns such as their difficulty with Theory coursework, the demanding pace with which the material was covered, and a lack of introductory level tools for exploration of FSMs.

Of the 18 students, 16 found jFAST easy to learn and use for creating FSMs. 15 students reported that they found the jFAST interface intuitive, and all 18 believed that using jFAST in conjunction with their Theory course would have improved significantly their understanding of FLA. General feedback on jFAST highlighted the usefulness of an FLA simulator for homework assignments to test and check automata. Interestingly, several responses suggested that jFAST might be used to provide students with the ability to view learning about FLA in terms of programming, a subject with which most computer science students are more comfortable.

4. CONCLUSIONS & FUTURE WORK

jFAST is a valuable and usable software tool to assist in teaching and learning about finite state machines, a concept some students find particularly challenging and initially impenetrable. Teachers and students found the software to be easy to learn and use, allowing the focus to be on the concepts rather than the software.

Contributions of jFAST include the use of a portable XML file format, enabling jFAST FSMs to be exported more easily to other applications, the use of a straightforward drag-and-drop interface, the colorful and unimposing graphical depiction of FSMs, a suite of example FSMs that support popular FLA textbooks, and a single, integrated tool for simulating all supported FSMs.

jFAST approaches the problem of FSM visualization by attempting to provide worthwhile educational software while minimizing the effort required to use the tool, a significant contribution to this domain. As a result, jFAST provides a complementary alternative to currently available software (such as JFLAP), focusing on students for whom the subject matter of FSMs is challenging or intimidating.

Although the feedback from the initial study is encouraging, the full value to Theory education that jFAST provides remains an open question. Further evaluation is planned through use in classrooms in the coming year. We plan to continue to refine the application as part of ongoing research at Villanova University's Applied Computing Technology Lab (actlab.csc.villanova.edu), and to extend its functionality with additional FSMs, the ability to import other popular FSM file formats, conversion between various machine types, and an expanded set of example FSMs from popular textbooks.

5. REFERENCES

[1] O. L. Astrachan, R. C. Duvall, J. Forbes, and S. H. Rodger. Active learning in small to large courses. In Proceedings of the 32nd ASEE/IEEE Frontiers in Education Conference, Boston, MA USA, November 2002.

[2] H. Bergström. Applications, Minimisation, and Visualisation of Finite State Machines. Master's Thesis. Stockholm University, 1998. Related website at: http://www.dsv.su.se/~henrikbe/petc/.

[3] W. E. Campbell and K. A. Smith. New Paradigms for College Teaching. Jossey-Bass Publishers, San Francisco, 1995.

[4] J. Bovet. Visual Automata Simulator, a tool for simulating automata and Turing machines. University of San Francisco. Available for download at: http://www.cs.usfca.edu/~jbovet/vas.html. 2004

[5] R. Cavalcante, T. Finley and S. Rodger. A visual and interactive automata theory course with JFLAP 4.0. In Thirty-fifth SIGCSE Technical Symposium on Computer Science Education, pages 99-99, ACM Press, 2004.

[6] N. Christin. DFApplet, a deterministic finite automata simulator. Available for download at: http://www.sims.berkeley.edu/~christin/dfa/. 1998.

[7] E. Head. ASSIST: A Simple Simulator for State Transitions. Master's Thesis. State University of New York at Binghamton. 1998. Related website at: http://www.cs.binghamton.edu/~software/.

[8] M. Mohri, F. C. N. Pereria and M. D. Riley. AT&T FSM Library. Software tools. 2003. Available at: http://www.research.att.com/sw/tools/fsm/.

[9] S. H. Rodger. Visual and Interactive Tools. Web site of automata theory tools at Duke University, Feb. 2005. Available online at: http://www.cs.duke.edu/~rodger/tools/.