A Java-based Tool for Reasoning About Models of Computation Through Simulating Finite Automata and Turing Machines

Matthew B. Robinson
WebScope, Inc.
3977 E. Bayshore Rd.
Suite 200
Palo Alto, CA 94303
650.965.2500
matt@webscope3d.com

Jason A.Hamshar
Sterling Software
Beeches Technical
Campus, Rt. 26N
Rome, NY 13440
315.336.0500
jason_hamshar@itd.sterling.com

Jorge E. Novillo
SUNY Institute of Technology
P.O. Box 3050
Utica, NY 13504-3050
315.792.7352
jorge@sunyit.edu

Andrew T. Duchowski
Clemson University
451 Edwards Hall
Clemson, SC 29634-1906
864.656.7677
andrewd@cs.clemson.edu

1. Abstract

Interactive visualization tools for models of computation provide a more compelling means of exploration and feedback than traditional paper and pencil methods in theory of computation courses. The Java Computability Toolkit (JCT) is introduced here as a new teaching aide and as an exploratory student's supplement to a course on theory of computation. JCT consists of two Java multiple-window, web-accessible, graphical environments, allowing the construction and simulation of finite automata and Turing machines. This paper discusses JCT's use, design, and applications in teaching.

2. Introduction and background

A new graphical simulation of finite automata (FA) and Turing machines (TM) is introduced, dubbed the Java Computability Toolkit (JCT). JCT is a multiple-window, web-accessible *program animator* [5] providing consistent appearance and functionality across platforms. The finite automata environment models all binary and unary closure operations and offers a unique circuit board representation of constructed FAs. The Turing machine environment utilizes a high-level notational specification providing hierarchical machine organization with locally or globally scoped variables. Incorporating both environments, JCT forms an integrated framework for web-accessible visualization of fundamental theory of computability concepts.

Machines in each environment contain specification and author information, allowing collection and distribution among students, colleagues, and classes at different universities. JCT can be used during lectures or laboratory sections to demonstrate the construction and function of various machines in which students are encouraged to participate.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGCSE '99 3/99 New Orleans, LA, USA
© 1999 ACM 1-58113-085-6/99/0003...$5.00

Theoretical models of computation are studied in the academic computer science community and often form the major portion of a core course in undergraduate and graduate degree programs. FAs and TMs are among the most common models in this discipline and, in order to fully develop a solid understanding of them, direct implementation and experimentation is necessary. This is usually undertaken with pencil and paper sketches which often results in messy, unmanageable machines that are difficult to follow, take a long time to create, and are virtually impossible to modify. Performing a computation on large machines is also a time consuming, sensitive task that can often lead to errors and frustration.

The ability to construct, modify, and perform simulated computations on machines may save students time and relieve some of this frustration. A graphical computer environment allows students to create machines interactively and to quickly respond to given feedback. Thus learning about models of computation is made more practical and allows a level of exploration and creativity not possible using traditional paper and pencil methods. This approach to visualization and interaction in computer science education has been used in the study of data structures [12], the introduction of language features and selected topics in introductory computer science [1], the construction of algorithms through animation [15], and, most relevant to this paper, in the study of formal languages and models of computation [3,13,16]. In [18] the role of such visualization techniques in computer science education is recognized as potentially useful for teaching and learning, especially where students are encouraged to learn on their own. Educational software is beginning to shift toward web-based design, simplifying installation and platform compatability issues, and providing immediacy of access. Examples and discussion of this shift can be found in [4] and [7].

There are a number of software packages for simulating finite automata and Turing machines. Most, however,

cannot be run through a web browser, are specifically designed to run on one platform, and require the user to install, and in one case, pay for the software. Turing's World [2] is a commercial package allowing the user to create TMs and FAs on the Macintosh. Both models are created in the same environment in which a state-based notation is used. This environment also allows the user to look into the computations of any sub-machine. Visual Turing (available at http://apolo.cs.pub.ro/~cheran/vturing/) allows the creation of TMs using a partial high level notation with multiple variables on a Windows platform. Deus Ex Machina [17] also runs on Windows and allows creation and simulation of seven different models of computation with a common state-based notation very similar to that of Turing's World. DynaLab, a finite state animator, is in the process of being ported to the web [5,6]. JFLAP (available at http://www.cs.duke.edu/~rodger/tools/jflap/) was the first powerful Java-based tool to allow simulation of FAs, push-down automata (PDAs), and TMs. The most recent version uses a state-based notation for all models, and allows conversion of a nondeterministic FA (NFA) to a deterministic FA (DFA), minimization of a DFA, and conversion of an NFA to a regular grammar.

JCT is a new multiple-window, Swing-based (see section 3) software package allowing construction of finite automata and Turing machines graphically on the computer. By providing arbitrary input, students can see how a machine works on that input, step by step, through immediate visual feedback. The use and features of JCT are discussed in the next section. Section 4 discusses the design of JCT and section 5 discusses future work and gives concluding remarks.

3. JCT

JCT takes advantage of the new Swing components (currently available at http://java.sun.com/products/jfc/) which allow true platform independent graphical user interfaces. These components are still under development and will be a core part of the new Java Development Kit, version 1.2, scheduled for first official release in late 1998. Previous to Swing, Java was limited to its dependence on AWT (Abstract Windowing Toolkit), which is a thin layer of code that maps to platform specific components at run-time. Thus, with AWT, a Java program appears and often functions differently from platform to platform. Swing components are written entirely in Java and have the same appearance and functionality no matter what platform they are used on. Swing components also bring to Java a much more powerful set of controls in which components can overlap and windows can contain sets of sub-windows. By leveraging this power, JCT allows users to build and perform operations on multiple machines in separate windows all contained within one parent frame, resulting in a high degree of organization both programatically and visually.

The ability to run JCT through a browser is paramount. Because the Swing components are so new, browsers have not yet been built to directly support them. The Java Plugin (currently available at http://java.sun.com/products/plugin/) provides browsers with an alternative Java virtual machine which has built-in support for Swing. If the user does not have the Plugin it is automatically installed before JCT is loaded.

JCT offers the following simulation features:

- Complete set of binary and unary FA closure operations resulting in new usable and re-configurable machines.
- Unique Java Automata Toolkit (JAT) [8,9] circuit board representation of FAs.
- High-level notational specification of TMs, with machine grouping and sub-machine nesting to any level, similar to that introduced in [10] and used in [11].
- Local and global variable scoping: a method of syntactically simplifying TM represention by allowing the storage of tape square contents to be unique to, or passed through from, sub-machines at any level.
- Signed applet web browser execution providing saving, loading, and printing functionality.
- A multiple-window, multiple-environment interface with an easily extendable, pluggable design.
- Movable transitions and transition groupings.

These features are discussed in 3.1, 3.2, and section 4.

3.1 Finite automata environment

The JCT finite automata environment uses a multiple window interface allowing the user to create machines graphically on a canvas by positioning states with single or multi-symbol transitions. Nonfinal states are blue and final states are red. The initial state is represented by a yellow arrow-head attachment. (JCT is intended to be used with a color display.) Transitions with common source and destination states are grouped together and can be automatically or manually placed by the user. Transitions on the empty string are represented by just a "_" similar to JFLAP.

Transition group boxes contain arrows that always point to the destination state. Each transition within a group is separated into its own box and can be modified or removed without effecting any others in the group. The working alphabet is implicitly defined by all transition symbols used in the current FA but can be manually specified by the user. Figure 1 shows an NFA accepting the language $\mathbf{a^*aa^*} \cup \mathbf{a^*bba^*}$.

Also shown in Figure 1 is the operations toolbar which allows, in order from left to right and top to bottom, rendering of the currently selected FA in the unique JAT circuit board diagram, conversion to a DFA, Minimization, Complement, Kleene Star, Intersection, Union, setting all transitions to manual positioning, and setting all transitions to automatic positioning. In cases where a binary operation is chosen, a dialog is displayed allowing the user to choose the second machine with which to perform the operation. The resulting machine of any operation is displayed in a new window within the environment, is completely modifiable (with the exception of the circuit board diagram), and can be used in subsequent operations. Machines are never modified as a result of performing an operation to allow comparison between input and output.

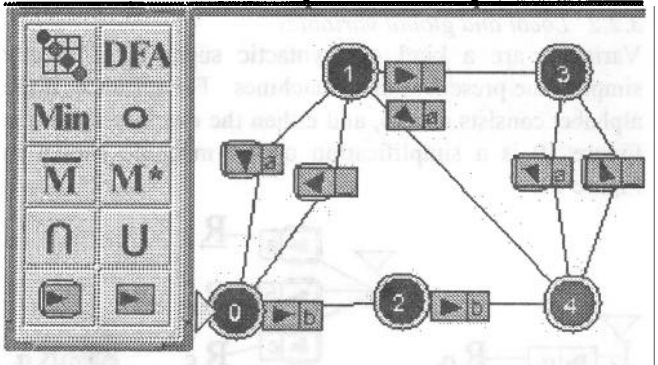

Figure 1: Operations Toolbar and Sample Finite Automata

The JAT circuit board diagram introduces a unique way of cleanly representing finite automata. Figure 2 shows the machine of Figure 1 in the circuit board diagram representation.

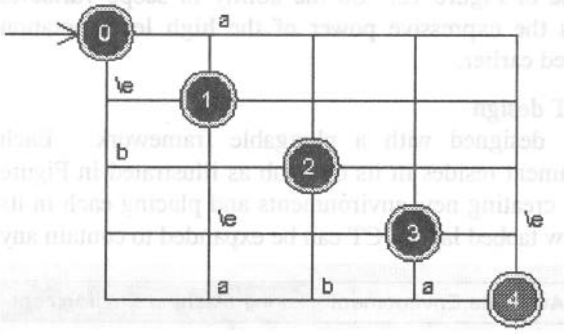

Figure 2: JAT Circuit Board Diagram Example

Although this is not a modifiable machine which can be used in operations or computations, it is very useful in organizing large automata into a more perspicuous format. Transitions from a low numbered state to a higher numbered state are placed below the diagonal and those from a higher to a lower numbered state are placed above the diagonal. Loop transitions are contained in their own sub-box. Transitions on the empty string are represented by "\e" in this diagram.

Given any FA the user can specify an input and JCT will perform the computation returning either "Accepted' or "Not Accepted". The computation will abort if a very high degree of nondeterminism is encountered. If an input is accepted, an optimum (shortest) acceptance path is displayed and the option to graphically trace through each step of the computation is presented to the user. Multiple traces can be simultaneously active allowing detailed comparison of machine functionality.

3.2 Turing machine environment

The Turing machine environment uses a multiple window interface and allows the user to create high level TMs graphically by organizing machines into groups on a canvas and creating transitions between them [10]. This is a hierarchical notation in which increasingly complex machines are built from simpler materials [11]. (Our notation is a unique version of that introduced in [10] and used in [11].

The user is presented with five basic machines and is able to incorporate any pre-existing machine as a sub-machine (e.g., one previously saved to disk). These are selected from a toolbar shown in Figure 3. The basic machines consist of move left, move right, move left until a specified symbol, move right until a specified symbol, and write a specified symbol. Each sub-machine is represented as a numbered icon. Figure 4 demonstrates how the user can be reminded of what a submachine is by placing the mouse over this icon (in this case a copy.TM is identified). Figure 5 illustrates the optional machine

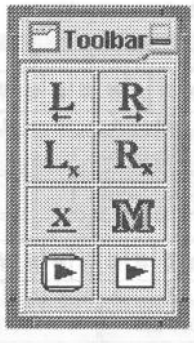

Figure 3: Machine Toolbar

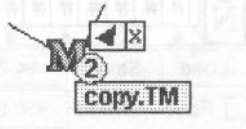

Figure 4: Sub-machine

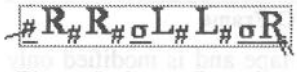
Figure 5: Group Boundary

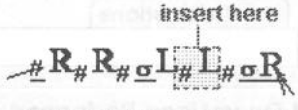

Figure 6: Group Insertion

group boundaries that can be toggled and Figure 6 shows how a machine can be inserted at any position within a group. The initial machine is represented by a yellow arrow-head attachment.

All transitions from one machine to the same destination machine are grouped together and can be manually positioned by the user, or automatically placed

by the environment (Figure 7 shows manually placed transitions and Figure 8 shows an automatically placed transition). Because the TMs created in this environment are deterministic, several rules are enforced by the environment to prevent the user from creating a nondeterministic Turing machine. Only one transition per symbol can be defined out of any machine on the canvas. A special transition called the "default" transition, also referred to as the "not" transition, is represented by a "_" and is defined to be a transition on all symbols except those explicitly defined out of the current machine. The default transition shown in Figure 7 is defined to be a transition on all symbols except "**#**" since there already is an explicitly defined transition on the symbol "**#**" out of the same machine.

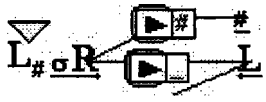

Figure 7: Default Transition

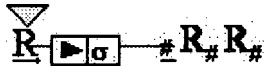

Figure 8: Variable Transition

A transition on a variable, shown in Figure 8, is defined to be the same as a default transition but it stores the tape symbol at the current head position into that variable. This variable can then be used in a write machine at any point to write the stored symbol to the tape. Variables and the distinction between local and global scoping is discussed in section 3.2.2.

3.2. 1 Tape input and output

The Turing machine environment tape I/O mechanism consists of two (practically) infinite two-way tapes. One

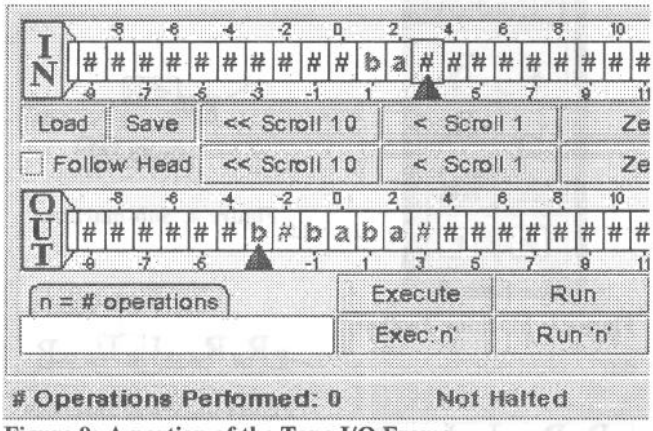

Figure 9: A portion of the Tape I/O Frame

tape is used strictly as an Input tape and is modified only by the user. When a computation is invoked the results are displayed in the second tape which is used strictly as an uneditable Output tape. Figure 9 shows part of the Input and Output tapes and some of their controls.

There are several different methods of performing a computation on a TM. The user can choose 'Execute' which will perform operations continuously until a halt state is reached or 500 operations are performed, whichever occurs first. If 500 operations are performed without reaching a halt state the user is prompted to continue or stop the computation.

The user can 'Run' a computation which will perform one operation every 0.5 seconds updating the Output tape as it proceeds. When toggled, the 'Follow Head' checkbox locks the user's view on the Output tape head to avoid losing site of it. A 'Trace' can also be initiated which allows the user to walk through a computation step by step.

3.2.2 Local and global variables

Variables are a kind of "syntactic sugar" that greatly simplify the presentation of machines. For instance, if the alphabet consists of a, **b,** and c then the machine shown in Figure 10 is a simplification of the machine shown in Figure 11.

Figure 10

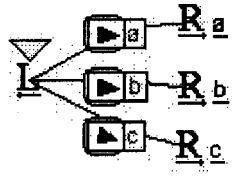

Figure 11

Figure 12

As a TM is constructed its variables may be locally or globally scoped. Global variables are used as a means of passing information to and from sub-machines. For instance, if alpha was declared global in the machine of Figure 10 then the machine shown in Figure 12, using the machine of Figure 10 as a submachine, obtains the contents of alpha from within the submachine. If alpha was declared local in the machine of Figure 10 this information would not pass through to the alpha in the machine of Figure 12. So the ability to scope variables extends the expressive power of the high level notation discussed earlier.

4. JCT design

JCT is designed with a pluggable framework. Each environment resides in its own tab as illustrated in Figure 13. By creating new environments and placing each in its own new tabbed layer, JCT can be expanded to contain any

Figure 13: Environment Tabs

number of self-contained environments. Not only does this aleviate the burden on the future developer, it also allows each environment to be used simultaneously without cluttering up the users' desktop with different frames.

Both the finite automata and Turing machine environments have distinct application (GUI) and library layers. The construction of the application layer is discussed further in [14]. An overview of the construction of the library layer that provides most of the operational algorithms for both environments can be found in [8] and [9].

5. Conclusions and future work

JCT contains environments for the construction and simulation of finite automata and Turing machines that can be used as a supplement to any theory of computation course. Its ease of use provides students with a powerful visual aid to learning and may equip students with a less frustrating and time consuming way to experiment with these models.

JCT has received positive feedback from faculty at SUNY Institute of Technology where it will be integrated into this year's foundations of computation course. A performance study of students using JCT will be held there as well.

A PDA environment is currently under development for future addition to JCT.

JCT is available at http://turing.sunyit.edu/JCT

6. References

[1] Astrachan, O. and Rodger, S.H. Animation, Visualization, and Interaction in CS 1 Assignments. In Twenty-ninth SIGSCE Technical Symposium on Computer Science Education, pages 317-321, 1998.

[2] Barwise, J and Etchemedy, J. Turing's World, Stanford: CSLI Publications, New York: Cambridge University Press. 1993.

[3] Bilska, A., Leider, K., Procopiuc, M., Procopiuc, Rodger, S., Salemme, J., Tsang, E. A Collection of Tools for Making Automata Theory and Formal Languages Come Alive. In Twenty-eighth SIGSCE Technical Symposium on Computer Science Education, pages 15-19, March, 1997.

[4] Boroni, C.M., Goosey, F.W., Grinder, M.T., Ross, R.J. A Paradigm Shift! The Internet, the Web, Browsers, Java, and the Future of Computer Science Education. In Twenty-ninth SIGSCE Technical Symposium on Computer Science Education, pages 145-152, 1998.

[5] Boroni, C.M., Goosey, F.W., Grinder, M.T., Ross, R.J., Wissenbach, P. WebLab! A Universal and Interactive Teaching, Learning, and Laboratory Environment for the World Wide Web. In Twenty-eighth SIGSCE Technical Symposium on Computer Science Education, pages 199-203, March, 1997.

[6] Boroni, C.M., Eneboe J.T., Goosey F.W., Ross, J.A., Ross, R.J. Dancing with DynaLab: endearing the science of computing to students. In Twenty-seventh SIGSCE Technical Symposium on Computer Science Education, pages 135-139, 1996.

[7] Cole, D., Wainwright, R., and Schoenefeld, D. Using Java to Develop Web Based Tutorials. In Twenty-ninth SIGSCE Technical Symposium on Computer Science Education, pages 92-96, 1998.

[8] Hamshar, J.A. Capabilities of the Java Automata Toolkit. Technical documentation, 1997. Available at http://turing.sunyit.edu/JCT/JATFA.htm (last referenced 8/24/1998)

[9] Hamshar, J.A. Turing Machine Capabilities defined within the Java Automata Toolkit, Technical documentation, 1998. Available at http://turing.sunyit.edu/JCT/JATTM.htm (last referenced 8/24/1998)

[10] Hennie, F.C. Introduction to Computability, Reading, Mass.: Addison-Wesley, 1977.

[11] Lewis, H., and Papadimitriou, C. Elements of the Theory of Computation, Second Edition, Prentice-Hall Inc., Upper Saddle River, New Jersey, 1998.

[12] Pierson, W.C. and Rodger, S.H. Web-based Animation of Data Structures Using JAWAA. In Twenty-ninth SIGSCE Technical Symposium on Computer Science Education, pages 267-271, 1998.

[13] Procopiuc, M. Procopiuc, O., and Rodger, S.H. Visualization and Interaction in the Computer Science Formal Languages Course with JFLAP. In 1996 Frontiers in Education Conference, Salt Lake City, Utah, pages 121-125, 1996.

[14] Robinson, M.B. A Java-based Tool for Models of Computation, Master's Thesis, SUNY Institute of Technology, July 1998.

[15] Rodger, S.H. Integrating Animations Into Courses. In ACM SIGCSE/SIGCUE Conference on Integrating Technology in Computer Science Education (Barcelona)(1996), pages 72-74.

[16] Rodger, S.H. Integrating Hands-on Work into the Formal Languages Course via Tools and Programming. Lecture Notes in Computer Science 1260, Springer-Verlag,. pages 132-148, 1996.

[17] Savoiu, N. Deus Ex Machina. Software accompanying Taylor, R.G. Models of Computation and Formal Languages, Oxford University Press: New York, 1997.

[18] Turner, A.J. Technology in Computing Education: Yet Another Bandwagon?. In ACM SIGCSE/SIGCUE Conference on Integrating Technology in Computer Science Education, Working Group Reports and Supplemental Proceedings, pages 121-124, 1997.