Finite State Machine Simulation in an Introductory Lab

Ryan McFall
Department of Computer Science
Michigan State University
East Lansing, MI
mcfallry@cps.msu.edu

Herbert L. Dershem
Department of Computer Science
United States Air Force Academy
USAF Academy, CO
hdershem@cs.usafa.af.mil

Introduction

TUMS (TUring Machine Simulator) is a program that we have developed to make the process of learning about finite state automata and Turing machines easier for the undergraduate student. It is our belief that allowing students to visualize the actions of such a machine will greatly enhance their understanding of how it works. TUMS provides a graphical interface that allows the student to construct and execute these machines. The software runs under the OpenWindows platform on a Sun SPARCstation. The motivation for the project arose from the use of a package called Turing's World (1986) which runs on the Macintosh platform. The intent of TUMS is to provide the functionality of Turing's World on SPARC architecture, as well as to provide some features not present in Turing's World.

Another Macintosh based program called Hypercard Automata Simulation has been developed at Union College by Hannay (1992).

This paper describes the TUMS software as well as its use in various courses in the Hope College Computer Science curriculum.

The Software

When TUMS is started the screen appears as in figure 1. Along the left side of the screen are the buttons that control the construction of a machine. The large window to the right of these control buttons is the "canvas" where the machine is built. Along the bottom part of the screen is the area where the input tape(s) is(are) displayed. Finally, the panel on the extreme right of the window simulates the action of a push down stack for those machines where one is present.

The first step in constructing any machine is to place the start state on the canvas. Since every machine requires a start state, TUMS will not allow anything else to be specified until the start state has been placed. Of course, it is possible that the start state might also be one of the final states; this can be accomplished by selecting the final state button before positioning the start state on the canvas. Positioning the state on the canvas is achieved by moving the cursor (which takes the shape of the object that is about to be placed) to the desired location and clicking the left mouse button once.

Once the start state has been placed, the user is free to continue building the machine in any order he or she wishes. The buttons on the left always control the "current action." For example, to place a final state, the user first selects the final state button and then places the state on the canvas.

Defining the transitions between the states is equally intuitive. First, the user selects the transition button. Clicking with the cursor inside of a state designates that state as the source state; a second click in any state chooses the destination state. After the destination state has been selected, a window like that in figure 2 appears. Depending on the type of machine being constructed, different fields in the panel will be active to allow the user to specify which action(s) should be taken for this transition.

After the machine has been completely defined, the next step is to specify the input string to be used. This is done by moving the pointer to the input tape in the bottom area of the screen. A character is put on the tape by clicking the left mouse button repeatedly until the desired character appears. The program will cycle through all available characters in the alphabet as the button is pressed. Note that it is possible to select any set of symbols for the machine's alphabet by pulling down the *Select Alphabet* menu and then selecting *Choose Alphabet* from the menu. TUMS defaults to having an alphabet of 'a', 'b', and '*'.

After the input has been specified, the user must select the starting position on the tape. TUMS reads from the input tape in a right to left manner. To designate the starting character, you use the mouse to position the arrow underneath the input tape below the desired start symbol.

Figure 3 depicts a simple machine with an input string specified, ready to run. First we will describe how a deterministic finite state automaton executes in TUMS. When the *Run Machine* button is selected, the machine begins its execution. The state that the machine is in at any given time is represented as a filled circle, while the input tape's arrow always points to the next character to be read. As the machine runs, the user is able to follow the path of execution by watching the highlighted state and the arrow.

There are two options available when running a machine. The first option determines how long the program waits before reading the next symbol from the input tape. The default value is approximately one second. By pulling down the *Options* menu and then selecting *Run Options*, you can either change the time interval (from a minimum of zero seconds to a maximum of two seconds), or specify that TUMS should wait for a key to be pressed before reading the next character.

The second option determines the way sub-machine execution is visualized. The default action is to have the current machine cleared and the sub-machine drawn on the canvas while it is executing. During this execution, the sub-machine behaves exactly as if it were a stand-alone machine. However, with an appropriate menu selection, sub-machines execute invisibly and instantaneously. That is, the "main", or "calling" machine stays on the screen and the sub-machine executes in the background, with no delays between consecutive main machine reads of the input tape.

SIGSCE 94- 3/94, Phoenix, Arizona,USA

When the machine completes execution, a window appears with information regarding whether or not the input string was accepted. If not, it contains a brief description of what caused the string to be rejected.

Non-deterministic machines operate slightly differently. Rather than forcing the computer to make a choice of which path to take when more than one possibility is present, a parse tree is constructed displaying all possible paths. At this time, TUMS only displays those paths that lead to the input string being accepted. In the next revision, all accepting paths will be displayed in black, non-accepting in gray.

TUMS can also run machines other than DFA's or NDFA's. Push Down Automata can be implemented, as well as one or two tape Turing Machines. We omit a discussion of these other machines for space considerations. The basic steps, however, are identical to those used in building a finite state automaton. In all cases, the user has the ability to save both machines and tapes to disk. The ability to print a copy of the state transition diagram is planned, but currently not implemented.

The Laboratory Exercise

During the fall semester of 1992 a new component was introduced in the introductory computer science class at Hope College. This is a two-hour weekly lab section, designed to give an introduction to some topics in computer science that are generally not presented to an introductory level computer science student. Topics covered included software testing and evaluation, modeling and simulation, cellular automata, and finite state automata. The last topic is where TUMS fits in.

Finite-state automata are difficult to understand, especially for students with little computer science background. The ability to visualize these machines and watch them perform through animation places the corresponding concepts within the grasp of the beginning student.

The laboratory was designed to give the students an introduction to the principles behind finite state automata, and then allow the students to explore these concepts in more detail on their own. An abbreviated copy of the laboratory is included as an Appendix to this article.

Upon starting the program, the students were asked to load some simple machines that we had previously constructed and experiment with them. Once they had determined what language each of them accepted, we then asked them to construct and test a couple of machines on their own.

The final exercise asked them to design an FSA that accepts input of the form $a^n b^n$. Of course, this is a classic exercise designed to demonstrate the limitations of finite state machines. After the students had tried various methods of doing so, we explained to them why such a machine was impossible, and demonstrated using TUMS a simple push-down automaton that could accomplish the task.

Evaluation

Students had fun working with the program and gained a genuine understanding of the concepts. Several of the students stayed after the regular class period to experiment with the software. The ability to test and redesign their machines gave them a sense of accomplishment when they had completed the lab.

In addition, some benefits were realized that we had not intended. One of these benefits deserves particular mention. Since TUMS was conceived and written entirely by undergraduates, the students in the introductory class got a chance to see the type of "fun" projects that can be attempted as their problem solving and programming abilities progress. This is important because students often believe that the only thing computers can do is solve the problems typically presented to introductory students. It is important to convey that computers can be used as educational tools.

The students enjoyed working interactively with the computer. After designing a machine, the students used TUMS to obtain feedback about whether or not they had achieved the designated purpose. The visual nature of the program facilitated group work and interaction. Each member of a group could see what was happening on the screen and figure out why the machine had or had not accomplished its intended purpose. At one time or another each of the students found themselves in the role of teacher, explaining to the other members of the group what had gone wrong. They appeared to feel comfortable using the software to demonstrate their ideas to the other group members.

Because of the lack of any further evaluation, it cannot be determined if the students fully retained the concepts that were intended. However, complete retention was not our goal. Rather, we hoped that using TUMS would allow the students to explore the ideas we were presenting and aid them in acquiring a fundamental understanding of those ideas. Based on the success the students experienced in completing the lab exercises, it appears this goal was successfully accomplished

The majority of difficulties we had with the lab came from the students' unfamiliarity with the OpenWindows environment. However, this did not prove to be a major hindrance to the learning process; once they got accustomed to using the mouse, they were able to operate the software with little difficulty.

The students did manage to uncover several bugs in the program. These bugs were generally caused by events that were unforeseen, such as the user repeatedly pressing the mouse button because they thought the program was not working. Occasionally the effects were severe enough that we would have to kill the process and have the students start over. By the second semester of use, most of these bugs had been ironed out.

Future Directions

There are many improvements and refinements to TUMS that are in our future plans. The most important of these include improving the software's ability to handle sub-machines and non-deterministic machines and improving the software's robustness.

The laboratory will continue to be taught in future semesters. For the most part, we were happy with the content of the lab and the reception it received from the students. The only change that may be necessary is to give the students an introduction to Unix and OpenWindows before the lab so that they can focus more of their attention on finite state automata.

It is also planned that the software itself will eventually become a major component of the Theoretical Computer Science class here at Hope College. This will take place as soon as sub-machines and non-deterministic machines are fully implemented. We would also like to add the capability of printing the state-transition diagram for a machine.

Acknowledgments

We would like to thank Hope College students Don Lingle, Jason Bomers, and Brett Folkert and faculty members Gordon Stegink and Mike Jipping for their contributions to this project. This work was partially supported by the Research Experiences for Undergraduates Program of the National Science Foundation, grant number CDA-9200118.

References

Barwise, J. and J. Etchemendy, "Turing's World: A Computer-Based Introduction to Computability Theory." Kinko's Academic Courseware Exchange, Santa Barbara, CA 1986.

Hannay, D., "Hypercard Automata Simulation: Finite-State, Pushdown and Turing Machines," *SIGCSE Bulletin* 24(2),55(1992).

APPENDIX

The Laboratory

II. The Finite State Automata Program

At this stage, hopefully you're comfortable enough with the OpenWindows environment to use a program written for it. Change to the directory **/home/smaug/mcfall** and double click on the icon **theory**.

First, we will load a couple of sample machines to see how the program works.

-- move to file menu and press left mouse button
-- when a window pops up with a list of file names,
select (left button again) the one named **a_star_b_star.mac** and select it

Each circle on the screen represents a state of the machine, and the arcs between them represent the transitions. Above each transition is the character on which that transition will be applied. The start state has a ">" in front of its circle, while a final state is composed of two circles. Note that a start state can also be a final state (as it is in this case).

Every machine can have its own "alphabet" of characters that it understands. By default, these machines have an alphabet that consists of the characters 'a', 'b', and '*'

To run a machine, you must specify an input string, or tape. Load in the input tape **tape1.tap** from the **/home/smaug/mcfall/lab** directory. Draw a picture of what the input tape now looks like:

Before we start, we must tell the machine where to start reading the input tape. The machine reads from right to left, so drag the arrow indicating the current position to the furthest non-empty square to the right and release it.

Now we are ready to run the machine. Find the **Run Machine** button on the left side of the screen and select it. Describe what happens:

Now try the tape **tape2.tap**. Also, go up to the options menu and select **Run Options**. From the pop-up window that appears, choose **Key Press** for the pause type and select apply. This causes the program to wait for a key press before each character is read from the input tape. Run the machine this way and again describe the results:

Now it is your turn to try and create some machines. Select the reset machine button on the left side of the screen. You can use this button any time you decide you don't like what you have and want to start over. Constructing a machine is fairly simple. The buttons along the left side of the screen control the current "action" you are performing. Note that no matter what you do, the first state that you construct will be a start state. If you wish to make the start state a final state as well, you must select the final state button before you place the start state.

For practice, we will construct the a^*b^* machine that we loaded in earlier. Since our start state was a final state, we need to select the final state button. Now move the mouse into the drawing area and press the left mouse button where you want the state to appear. If the state you've drawn doesn't look the way it should, you might have moved the mouse too fast while selecting the final state button. If so, reset the machine and try again.

We need one other state, which is also a final state. You don't have to select the final state button again -- the "action" stays the same until you press another button. Place this state on the diagram in the same way as the first state.

To construct the transitions, first select the **transition** button. We want our first transition to be from state 0 to itself. Move the cursor inside state 0 and click the left mouse button twice. A window will pop up asking you to type in the character on which the transition occurs. Move the cursor inside this window and enter an 'a', then select the **Done** button. In the same way, add a transition from state 0 to state 1 on a 'b' and one from state 1 to itself on a 'b'. The machine is now complete.

If you wish to modify the diagram you can do so by first selecting the **state** button. Then pointing the mouse inside the state you would like to move, holding down the left mouse button and moving the mouse until the state is where you want it. Then release the mouse button. You can also change the curvature of a transition this way by dragging the label of the transition.

Now, try to construct machines that accept the following strings:

--a^+b^* -- ab^*a -- a^nb^n

Draw pictures of the machines you designed for the first two: If you've had trouble with the last one, there is a good reason for it. This string cannot be recognized by *any* finite state automaton. The reason is that you have no way of making sure that the same number of b's occur as a's.

Load in the machine **an_bn.mac** and the tape **tape3.tap**. Set the machine options delay type to **Key Press** and run the machine.

Keep an eye on the right side of the screen as the machine runs, and see if you can explain what is happening. Try some other strings and see what happens with those. You will see that this machine accepts exactly $a^n b^n$. Can you explain how it does it?

This type of machine is called a push-down automaton because of the presence of a push-down stack. If you continue on in Computer Science, you will encounter the stack data structure again many times.

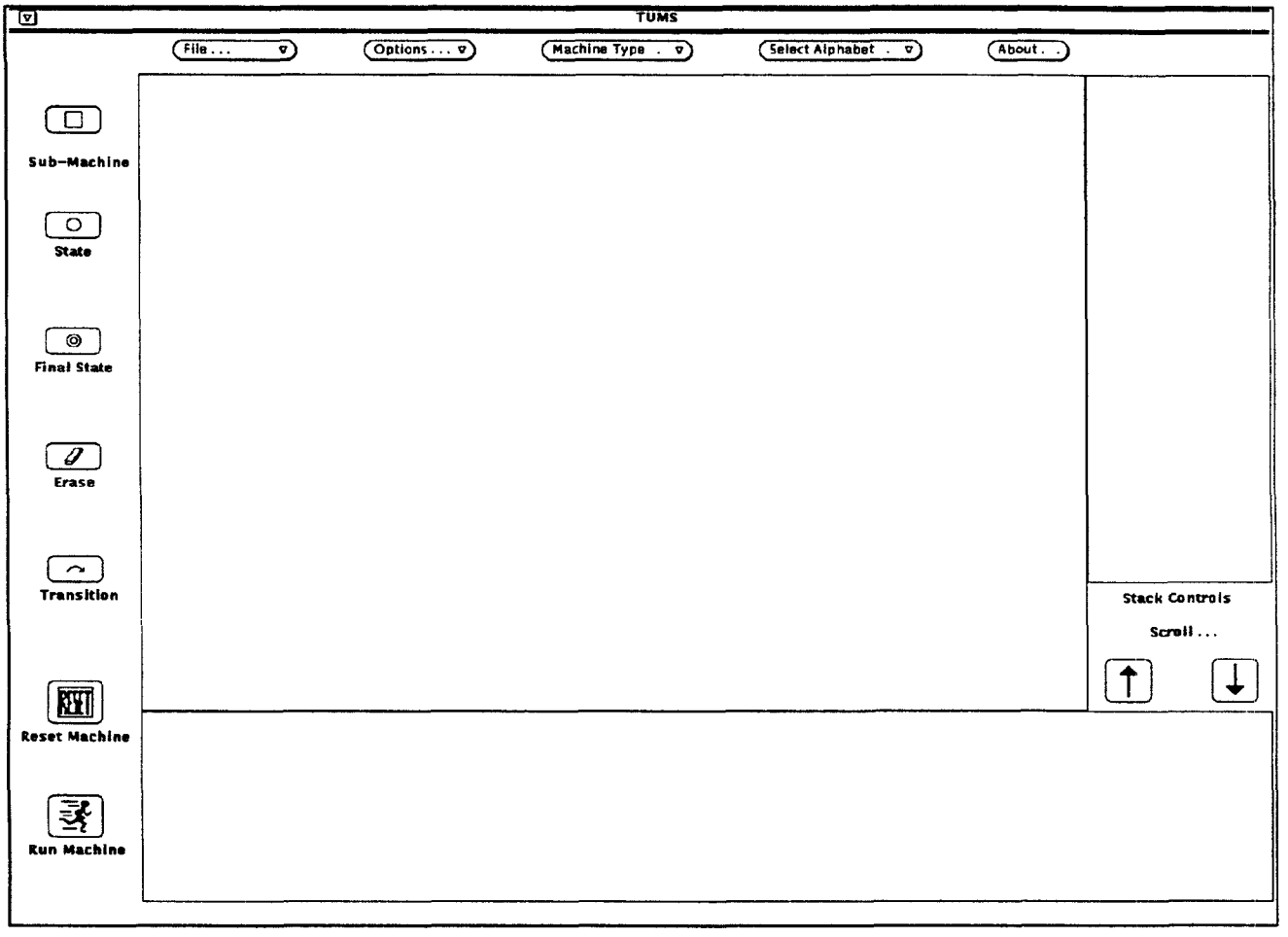

Figure 1: TUMS initial screen

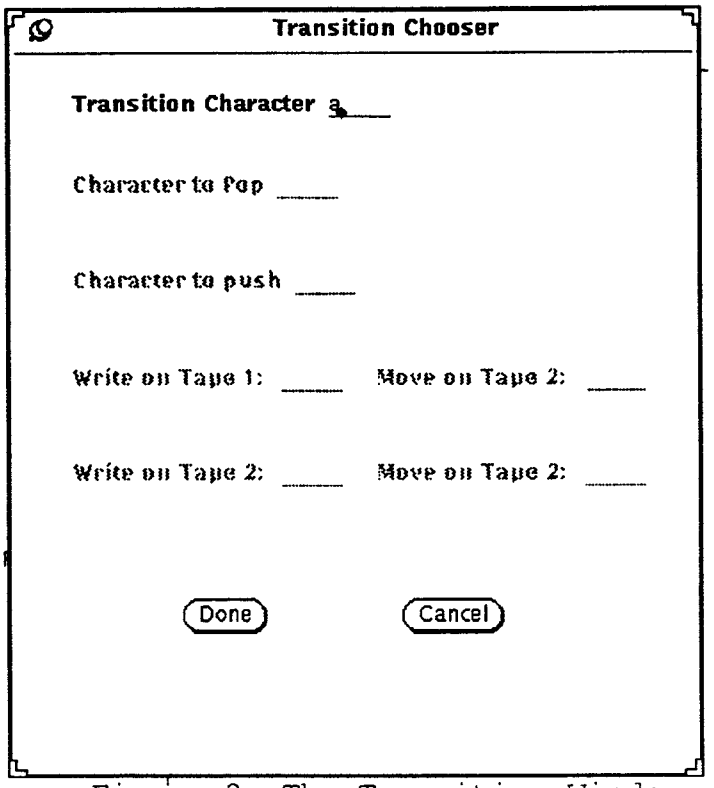

Figure 2: The Transition Window

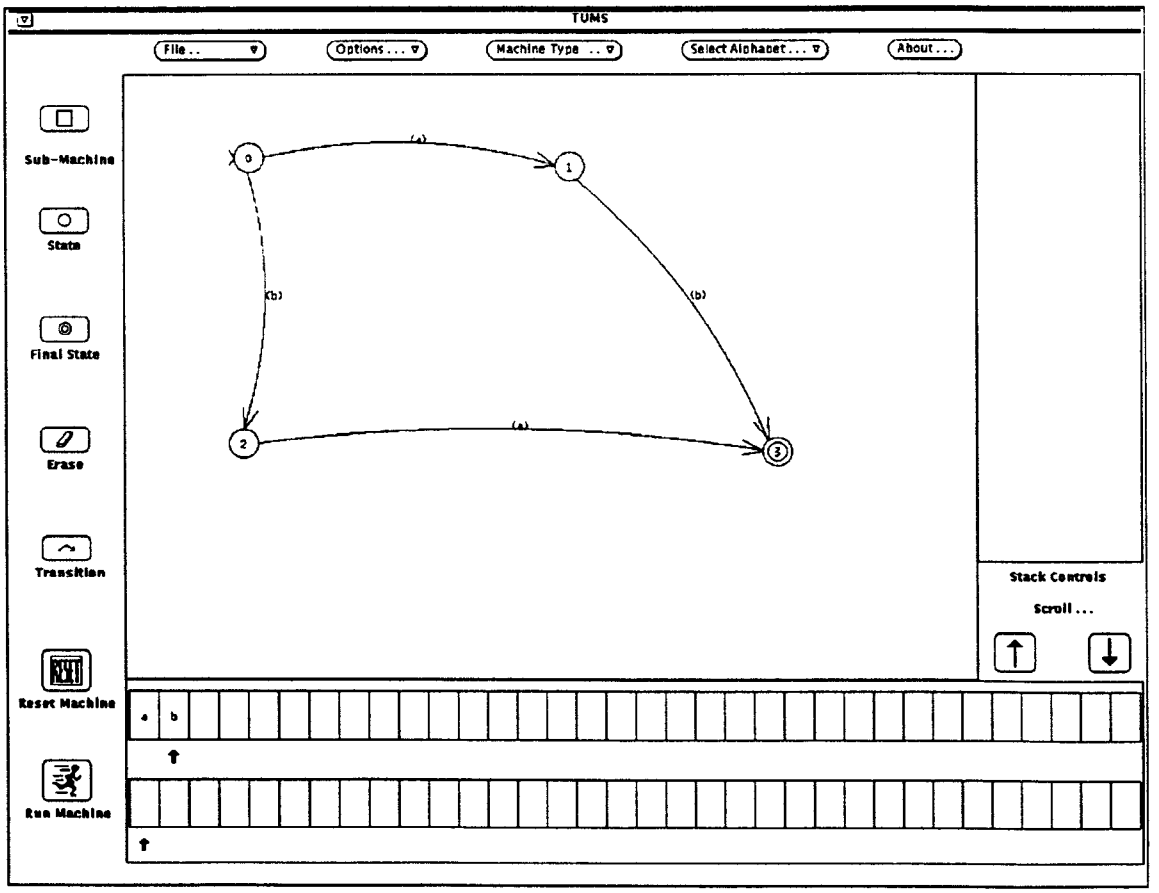

Figure 3: Simple Machine and Input Tape