A Visual Programming Environment for Turing Machines

Eric Luce and Susan H. Rodger
Computer Science Department
Rensselaer Polytechnic Institute
Troy, NY 12180-3590

Abstract

Visualization and interaction are powerful tools for explaining abstract concepts. We have developed TuBB, a visual and interactive tool for modular design and animation of Turing machines. Unlike previous visual tools for automata that are based on state diagrams, the Turing machines in TuBB are composed of previously defined Turing machines called building blocks. A Turing machine can be pictorially designed, stored in a library, and used as a subprogram to construct more complex Turing machines. In addition, TuBB brings the user constructed machine to life, providing simulations at varying speeds.

1 Introduction

Using visualization and interaction on theoretical concepts bring them to life, making them easier to understand. Visualization provides an alternative view that people can process more easily than textual information. Interaction allows one to experiment and verify their ideas. For example, animation systems, such as Zeus [2] and Tango [8], provide animated interpretations of algorithms and data structures, giving the user an alternative view to aid in understanding and letting the user participate by selecting the input and controlling the speed of the animation. Tools have also been developed [1, 6, 9] for visualizing and interacting with formal models of machines, which are important because they form the foundations of computer science, yet are difficult to understand because they have traditionally been presented in an abstract manner. These tools allow the user to draw an automaton pictorially, providing a useful aid in understanding the formal definitions of automata. However, they are limited to constructing small examples.

We have developed a visual and interactive tool for designing and simulating Turing machines that allows the user to construct large modular examples and aids in understanding the computational power of a Turing machine. Our tool, named TuBB (Turing Building Blocks), provides an environment for constructing Turing machines that are built out of previously defined Turing machines, called building blocks. Using TuBB, one can interactively draw a Turing machine by selecting and connecting building blocks stored in a library. The machine constructed is a new building block that can be named, stored in a library, and used to construct more complicated machines. In addition, TuBB brings the user constructed machine to life, providing a simulation at varying user controlled speeds.

In Section 2 we describe in more detail previous work on interactive tools for automata. In Section 3 we define the basic building blocks and describe how they can be combined to construct additional building blocks. In Section 4 we describe the user interface and implementation, in Section 5 we briefly show the relationship of the major components of the implementation, and in Section 6 we state some concluding remarks.

2 Previous Work on Automata Tools

Many of the previous tools for constructing and simulating Turing machines and other automata are textual based making them difficult to use in designing the automata. Sutner [10] developed a tool built on top of Mathematica for experimenting with finite automata that are represented as a tuple of sets in one long string of text. Both Hannay [4] and Dogrusoz [3] developed tools using Macintosh Hypercard stacks and X windows, respectively, for simulating several types of deterministic automata that are represented in tabular format. In all these tools, the textual representations of the automata do not pictorially represent the relationships between states, making it difficult to design the automata. Furthermore, since the textual representations of simple automata are quite lengthy,

1049-2615/93 $03.00 © 1993 IEEE

these tools involve typing in a lot of text.

There are several tools for designing and simulating automata that represent the automata by directed graphs, yet these tools are limited by the size of the graphs. AMS [5] is a tool for experimenting with several versions of finite automata, Turing's World [1] is a tool for experimenting with Turing machines, GDR [9] is a graph package that includes an example of finite automata, and FLAP [6] is a tool for experimenting with nondeterministic finite automata, pushdown automata and Turing machines. All these tools allow the user to construct pictorial representations of automata as state diagrams (labeled directed graph), making it easy to see relationships between states. However, these tools are limited because a very simple example usually contains a large number of states and arcs. For example, a Turing machine to make a copy of a string formed of the letters a and b contains 10 states and 30 arcs.

3 The Turing Machine Programming Language

In this section we define a Turing machine, informally and formally, and then describe how to construct a Turing machine composed of building blocks. A Turing machine is a powerful, yet simple machine. It consists of a tape, a tape head, and a control program. The tape, containing an infinite number of cells in which symbols can be read from or written to, represents the memory of the machine. The control program executes a sequence of steps, in which each step is decided based on the current state and the current symbol. A step consists of reading the current symbol on the tape, writing a symbol, changing state, and moving the tape head one cell to the left or right.

Definition: A deterministic Turing machine M can be represented by a sept-tuple, $M = \{Q, \Sigma, \Gamma, \delta, B, q_0, F\}$, where Q is a finite set of states, Σ is the input alphabet, Γ is the tape alphabet (with $\Sigma \subseteq \Gamma - \{B\}$), δ is a function represented by $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, L\}$, B is a special symbol denoting a blank on the tape, q_0 is the start state ($q_0 \in Q$), and $F \subseteq Q$ is a set of final states. The symbols R and L denote the directions Right and Left.

In the formal definition, the function δ defines the transitions. An input string on the Turing tape is accepted if when started at the start state, transitions are processed and a final state is reached.

The example in Figure 1 shows a Turing machine represented as a transition state diagram. The tape

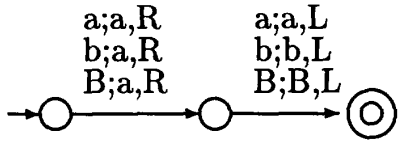

Figure 1: State diagram for "write a"

alphabet is defined to be $\{a, b\}$, a state is denoted by a circle, the start state is denoted by the small arrow, the final state is denoted by a double circle and a transition is denoted by a labeled arc. There are actually three labeled arcs between pairs of states, but we have only drawn one arc, with three labels. This program writes the symbol a on the current tape cell, and leaves the tape head pointing to this cell. Each transition has three parts: i;j,M. This is interpreted as: if i is the current symbol on the tape (under the tape head), replace it by writing the symbol j, and move one tape cell in the direction M. Thus the program simply writes an a on the tape cell, moves right one cell (in the definition, the tape head is always moving), moves left one cell (back to its original position), and halts. Writing a symbol is a complicated task involving three states and six transitions.

We now describe a method of naming and combining Turing machines into building blocks [7], each of which is a Turing machine. First, we define the basic building blocks.

L	-	move left
R	-	move right
x	-	write symbol 'x'

'L' denotes a Turing machine that moves left one tape square, 'R' denotes a Turing machine that moves right one tape square, and 'x' (where x is any symbol in the tape alphabet) denotes a Turing machine that writes the symbol x on the current tape square, and doesn't move the tape head. Each of these building blocks can be represented by a state diagram of a Turing machine (the state diagram for writing the symbol 'a' is shown in Figure 1), but our building block notation is simpler, just a single letter.

Building blocks can be combined easily in several ways. Each building block can be represented by a black box, showing only the start state and the final state of its state diagram. Two building blocks A and B can be combined sequentially into a new building block labeled AB or A→B by making the final state of A a nonfinal state, and adding a link (new state and

transitions) between the old final state of A and the start state of B. The new building block executes A first, and then executes B, with the tape head starting for B exactly where it left off for A. Building blocks A and B can also be combined in a conditional manner. Suppose A executes first. Upon A's completion, if the symbol 'a' is the symbol under the tape head, then execute B. This conditional combining is denoted as $A \xrightarrow{a} B$.

Negation can be applied to a symbol to mean any symbol except that symbol. An example of combining two building blocks A and B using negation and a condition is the following. A executes first. Upon A's completion, if the symbol 'a' is not the symbol under the tape head, then execute B. This combining is denoted as $A \xrightarrow{\neg a} B$.

The following are additional examples of Turing machines.

R_a	move right to next symbol 'a'
$R_{\neg a}$	move right to first symbol not an 'a'

We now show an example of combining building blocks to create a Turing machine. Figure 2 shows a Turing machine named C for copying a string of 1's. If the initial string on the tape is 111$\underline{B}$ (where the _ denotes the position of the tape head and B denotes a blank), then after the program completes, the tape will contain 111B111$\underline{B}$. Figure 3 shows a Turing machine for multiplying two unary numbers that uses the copy machine (shown by C) by making a copy of the second number for each 1 in the first number. If the initial string is 11B1111$\underline{B}$, then after the program completes, the tape will contain 11111111$\underline{B}$.

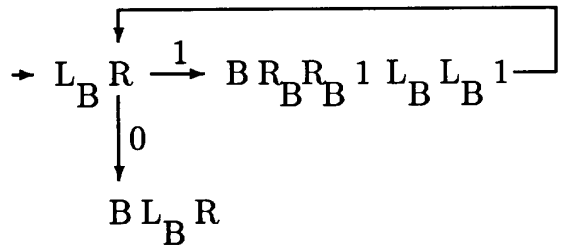

Figure 2: A copy machine named C

4 Overview and The User Interface

TuBB is written in C++ 2.1 using Unidraw [11], the InterViews 3.1 based toolkit for building domain

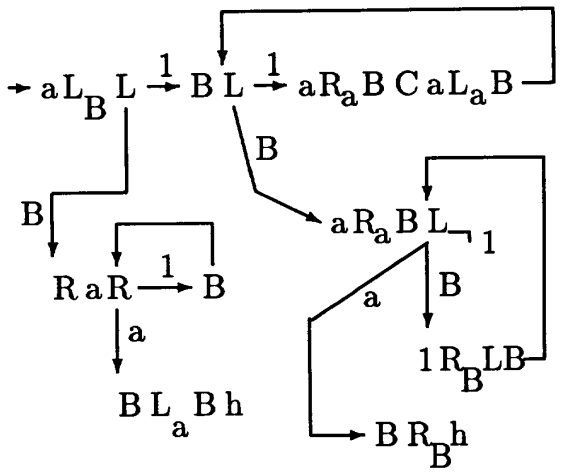

Figure 3: A multiply machine using C

specific graphical editors. The user interface consists of five parts: the Turing machine editor, the Turing machine library, the message window, the tape, and the run control area. These parts are shown in Figure 4.

4.1 The Turing Machine Editor

The graphical editor provides a visual environment for constructing a Turing machine program. It provides visual feedback while performing such operations as creating transition links, creating components of a Turing machine, deleting components, copying, cutting, pasting, and undoing.

The editor is made up of several smaller parts. The *display area* displays the Turing machine that is being constructed. The *menu bar* contains various menus for basic file operations, printing a postscript representation of the current program, editing commands, viewing characteristics, accessing the library, and miscellaneous settings. The *toolbox* contains tools for building and modifying a Turing machine. The *panner* and *zoomer* allow the view of the Turing machine to be shifted and zoomed so that the entire machine can be seen.

4.1.1 The editor's display area

The editor's display is a large area, initially empty, where the user visually constructs the Turing machine program. An example of the multiply machine described in the previous section has been constructed

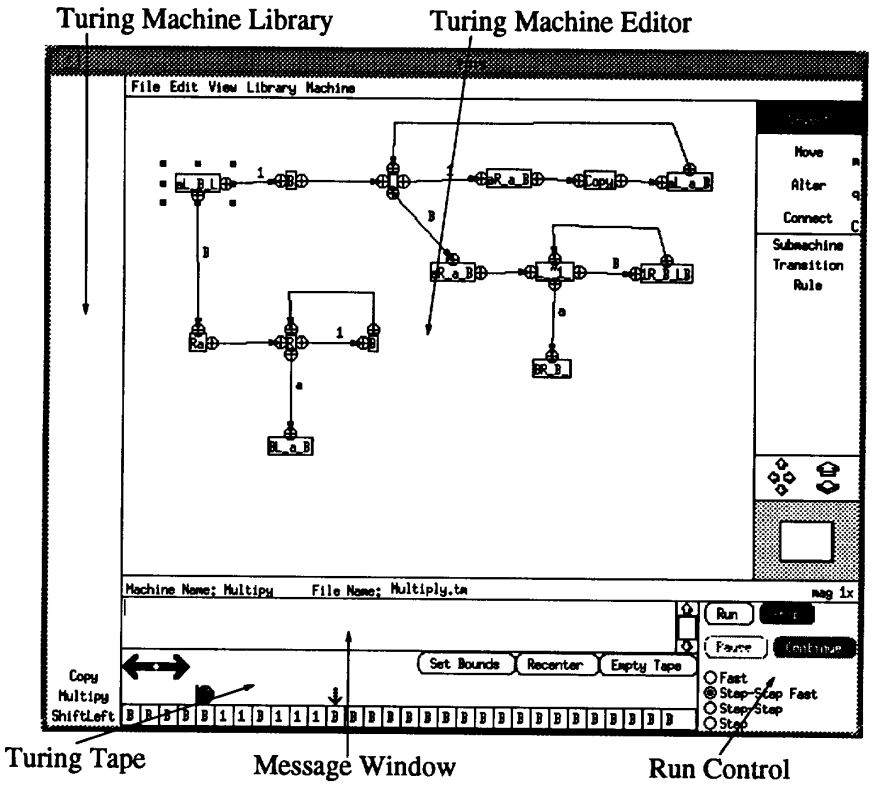

Figure 4: The Turing Machine Simulator

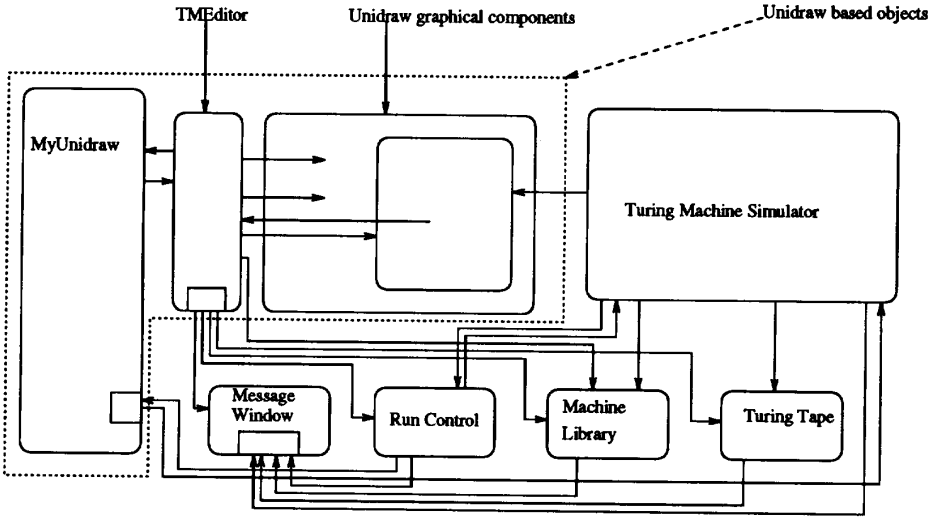

Figure 5: The major parts of the Turing Machine simulator's implementation

using TuBB in Figure 4. Pins, which can be made to disappear, show how all the components are glued together. The display area is actually larger than what can be viewed at once. The panner and zoomer are used to control what is visible at any one time.

Within the display area the various mouse buttons have different meanings. The **left** mouse button is used to invoke the currently selected tool from the toolbox or the library. The **middle** mouse button will temporarily invoke the move tool on the component that the pointer is over. When the middle button is released the selected tool will revert to the tool previously selected before the middle button was pressed. The **left** mouse button will temporarily invoke the select tool and select the object below the pointer. When the right mouse button is released the selected tool will return to the tool previously selected.

4.1.2 The Tool Box

When one presses the left mouse button in the editor's display area the currently selected tool is engaged and highlighted in either the toolbox or the library. In Figure 4, the toolbox is the rightmost column.

The toolbox is divided into two areas. The first area contains generic editing tools, and the second area contains a tool for each possible component of a Turing machine. The generic editing tools are: **Select** to choose a component or set of components, **Move** to move a group of selected components, **Alter** to edit a specific component, **Connect** to link two components which can be connected (like a transition to a submachine, or a rule to a transition.)

The tools used to create the components of a Turing machine are the following. The **submachine** tool creates an instance of a submachine. The default submachine created is for the L machine. The user may alter the default with the **alter** tool to invoke some other submachine. The **transition** tool is used to create transitions between submachines. The **rule** tool is used to create conditions on the transitions.

4.2 The Turing Machine Library

In order to provide the ability to construct complex programs, one of the features of TuBB is the ability to execute other Turing machine programs, or submachines. In order to do this we provide a "library" of programs. The user may use a program from this library directly in the Turing machine program they are currently designing.

The Turing machine library is really another toolbox similar to the one described as part of the Turing machine editor above. When the Turing machine simulator is started the Turing Machine library searches along a series of directories for any files which may contain Turing machines. Any Turing machines found are added as tools to the library. The user may then select one of these tools and it will behave exactly as the **submachine** tool in the toolbox of the Turing machine editor section. When the user applies this tool, however, instead of the L machine appearing in the label of the submachine component, the machine named in the library will appear instead.

4.3 The Message Window

The message window displays informational and error messages to the user during the building and simulating of their Turing machine. The message window consists of a display area and a scrollbar. Typically most messages displayed will occur during simulation when the Turing machine reaches an error, or halts. The scrollbar can be used to go back and look at previously displayed messages.

4.4 The Turing Tape

Being able to create and edit a machine online is of little use unless one can also verify that it works. Watching the machine as it steps through its states and modifies the tape can help the user debug and better understand how a Turing machine works. In order to provide this simulation environment we need two more components: a tape and controls for execution.

The tape consists of a series of cells. Each cell contains either one character from the alphabet or a blank. There is a tape head positioned at one of the cells on the tape. The tape is potentially infinite in both directions. We provide controls that allow for the machine to be halted, with the option of continuing, if the head passes an upper or lower boundary. This provides a safety check to identify what is most likely an infinite loop. The lower bound can be used to simulate a Turing tape which is infinite in only one direction.

The representation of the tape consists of four parts. The first part is the tape itself. This is represented as a series of boxes with a single character per box. The second part, above the tape, consists of the tape head and the upper and lower boundaries of the tape (shown as stop signs). The tape head is represented by an arrow pointing down to the current cell. The third part is a set of arrows which allow the user to scroll the tape either to the left or right.

The fourth part consists of push buttons that allow the user to set the upper and lower bounds, recenter the tape, and empty the tape.

4.5 The Run Control

We provide the run control area so that the user will have some way of starting, stopping, pausing, continuing, and setting the running speed of the simulation of their program. The pause feature allows the user to stop the simulation in order to modify the Turing machine or the contents of the tape. All of the controls except the speed controls appear as toggle buttons.

The speed control provides four different kinds of simulation modes. For instance, the user may want to have the simulator pause before executing a simulation of the Turing machine, waiting for the user to tell the simulator to continue. Alternatively, the user may want the machine to run continuously, with a short pause between executing all of the various actions, highlighting the components of the Turing machine displayed in the editor's display area as the machine is simulated. As another possibility, the user may want the machine to run to completion as quickly as possibly not taking time to pause between the steps of the machine being simulated, and not updating the image of the machine displayed in the editor's display area.

5 Application Implementation

The components of the implementation consist of six major parts: The Unidraw editor, the Turing machine library, the message window, the run control area, the Turing tape, and the simulator. Their relationships are shown in Figure 5.

6 Conclusion

We have developed an interactive tool, TuBB, for designing a picture of a Turing machine that then comes to life, simulating the running of the Turing machine on some input. TuBB allows modular design, using user defined Turing machines as building blocks, which allows the user to construct Turing machines that perform interesting tasks such as copying, multiplying, and sorting. This would be difficult and tedious to attempt without the use of the library. In addition, Tubb's step-by-step simulation aids in debugging.

Tubb can be used as an aid in teaching Turing machines in several ways. In the classroom the instructor can interactively design an example, with help from the class, and immediately show that it does or does not work. In labs, or outside of class, students can design different small tasks to store in the library, which can then be brought together to construct one large task. These examples show how our tool can be used to enrich the study of Turing machines.

References

[1] J. Barwise and J. Etchemendy, Turing's World, Kinko's Academic Courseware Exchange, Santa Barbara, CA, 1986.

[2] M. Brown, ZEUS: A System for algorithm animation and multi-view editing. *Proceedings of the IEEE 1991 Workshop on Visual Languages*, p. 4-9, Kobe, Japan, Oct. 1991.

[3] U. Dogrusoz, Abstract Formal Machine Simulator, Computer Science Department Master's Project, Rensselaer Polytechnic Institute, 1991.

[4] D. Hannay, Hypercard Automata Simulation: Finite State, Pushdown and Turing Machines, *SIGCSE Bulletin*, 24, 2, p. 55-58, June 1992.

[5] M. C. Lee, An Abstract Machine Simulator, *Third International Conference on Computer Assisted Learning*, p. 129-141, 1990.

[6] M. LoSacco, and S. H. Rodger, FLAP: A Tool for Drawing and Simulating Automata, *ED-MEDIA 93, World Conference on Educational Multimedia and Hypermedia*, June 1993.

[7] H. Lewis, and C. Papadimitriou, *Elements of the Theory of Computation*, Prentice-Hall, 1981.

[8] J. Stasko, Tango: A Framework and System for Algorithm Animation, *IEEE Computer*, p.27-39, Sept. 1990.

[9] M. Stallman, R. Cleaveland, and P. Hebbar, GDR: A Visualization Tool for Graph Algorithms, North Carolina State University Technical Report TR-91-27, (1991).

[10] K. Sutner, Implementing Finite State Machines, *DIMACS Workshop on Computational Support for Discrete Mathematics*, 1992.

[11] J. M. Vlissides and M. Linton, "Unidraw: A Framework for Building Domain-Specific Graphical Editors", Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology, 1989.