

Interactive Pushdown Automata Animation

Jennifer McDonald

College of Computer Science

Northeastern University

Boston, MA 02115

jenimac@ccs.neu.edu

Abstract

This paper will present the Interactive Pushdown Automata Animation for use in an Automata Theory class. It will present the features of the IPAA as well as the algorithm and data model used. Finally, this article will outline the necessary pieces of a good visual tool and show how they are implemented in the IPAA.

1 Introduction

As an undergraduate student studying computer science, I can honestly say that learning algorithms can be a daunting task. It is customary for professors to conduct lectures in front of a chalkboard outlining algorithmic methods and drawing diagrams. However, when it comes to complex algorithms, students often have difficulty fitting the pieces of the algorithm together. These gaps in conceptual understanding can prevent students from fully understanding the algorithmic process. In order to remedy the problem, educators need to provide hands-on visual tools in the classroom.

In this paper, I will introduce a blueprint for proper visualization tool design. I will then present a visual tool we designed to aid in an Automata Theory class. I will discuss the design features of the application along with the object-oriented lessons learned in the process of designing it.

2 Recipe for Visualization Tool Design

In order to create a tool that will not only capture student's interest but also help them learn, you must provide the following:

This work was partially supported by NSF grant DUE-9950829.

- **A Facility for Interactivity**

Students should be able to enter and test their own data. They should also be able to modify existing test data. If students are able to modify pieces of the puzzle and witness the results, it will better their understanding of the algorithm itself.

- **Multiple Representation of Data**

Not all students are created equal. What may be crystal clear for one may be incredibly cloudy for another. Providing multiple views of the data processing in the algorithm ensures that all students will benefit from the tool. This also helps those students who need to see the process a hundred different times in a hundred different ways until it "clicks."

- **Algorithmic Animation with Control**

All visual tools should provide a simulation of the process they are trying to teach. However, students should have control over the speed of the animation. If they see too many whiz-bang flashes of information zooming across the screen, they are likely to glaze over and not be any clearer than when they started. Likewise if the student has to sit through three minutes of slow-as-molasses animation, boredom will set in. Giving the students control of the animation allows them to work at their own pace, starting slowly and working faster as the algorithm becomes more obvious.

Ultimately the student should be able to stop the simulation mid-process in order to observe the current state of the data. An option to pause between each step of the process is often appreciated by students.

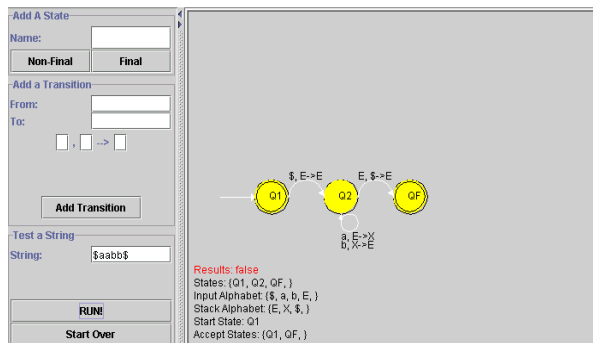
- **A Starting Point from which to Explore**

Providing an initial set of test data will help students acclimate themselves with the tool. They should be able to run the simulation a few times first to get comfortable and then they should be able to modify the given data and begin the exploration.

3 History of the Interactive Automata Project

The Interactive Pushdown Automata Animation tool was originally created as an honors project for my Automata Theory class. I was asked to create an interactive simulation for the automata of my choice. I opted to simulate pushdown automata because I felt it was the most interesting of all the automata we had learned in class.

I decided to write my program in Java using the Swing Interface. Unfortunately, creating a user interface with Swing proved to be far more difficult than I anticipated. I spent so much time debugging my GUI that I ran out of time and was unable to implement the majority of the algorithm to run the simulation.

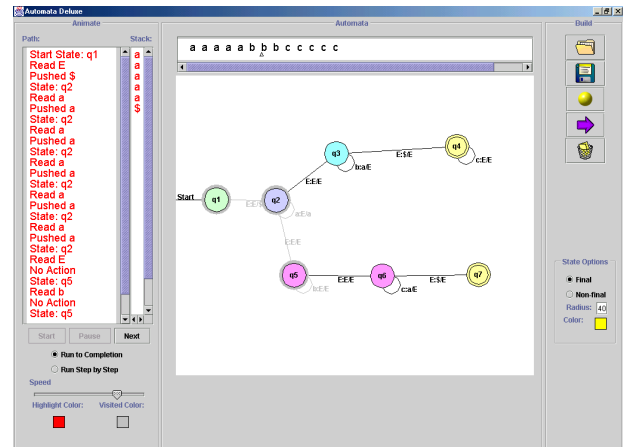


Early Pushdown Automata Program

Ultimately I ended up with a small program that allowed the user to create a non-deterministic automaton (with a maximum of five states). I was unable to implement the stack and there was no animation. The student could enter input strings to test the automaton, but the only feedback given was a message indicating whether or not the string was accepted. Needless to say, I was very disappointed with the outcome of my project.

A year later I was given the opportunity to work on a research project at my university. Under the direction of Richard Rasala, Viera Proulx, and Jeff Raab, I was able to explore the Java Power Tools as another honors project. The JPT is a toolkit designed to allow students to rapidly and easily develop graphical user interfaces. For more information on the JPT, please see [1].

After exploring the toolkit, it was clear to me that I finally had the means to complete the pushdown automata project. With the help of my mentoring professors, I completely re-wrote the entire application. By the end of the term, I found that I had gone above and beyond my original goals for the project. I had a full-featured application that allowed the user to explore pushdown automata in-depth with an easy-to-use interface. At this point, I realized that I had created a tool that could be a great help in the classroom.



Interactive Pushdown Automata Animation

4 The JPT and GUI Design

It is important that all controls and views fit on a single screen so that the UI is as simple as possible. The JPT came in handy at design time when it was necessary to swap pieces around to make everything fit in the appropriate places. The Display class in the JPT was crucial in laying out the components in a sensible manner. The Display class allows the designer to add titles and annotations in the desired places and also keeps components in their proper places when the window is resized.

The biggest advantage gained from using the JPT is the connection between the model and view. All JPT display components allow the designer to request a model from the view and set the view from a model. This symmetry allows the GUI to easily convey the changes made to the underlying data model when the algorithm runs. Without the convenient model-view connection of the JPT, this project would not have been possible.

Now I will demonstrate how the IPAA provides all of the features outlined in the recipe for visual tool design outlined earlier in the paper.

5 Facilitating Interactivity

The controls to the right of the screen enable a student to construct his or her own automata. Using the iconified buttons, students may:

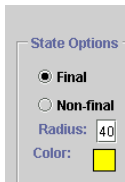


- Load an automaton from a file
- Save the current automaton to a file
- Add a state
- Add a transition
- Clear the automata window along with the underlying data structure.

The entire application is mouse-driven so the student may add states to the window without knowing the coordinates of the window at that position. Likewise the student may select the source and target states for a transition by clicking on the desired states. The transition symbols themselves are input via the keyboard.

When entering the symbols, the student should follow the notation described by Sipser [2]. For example: the transition symbols $a:E/X$ indicate that the transition moves when an “a” is read from the tape, and an “X” is pushed on the stack in place of an epsilon. Note: in this design, the character “E” is reserved to represent epsilon.

When adding a state to the automata window, students control the following:



- Whether the state is final or non-final
- The size of the state
- The color of the state

The final/non-final options are presented as a JPT OptionsView[1]. The radius is presented as a JPT TextFieldView[1] with built-in error checking and expression evaluation. For example, one could enter the string “3 *pi” in the radius field and the application would use the appropriate numerical value for the state radius. The color is a JPT ColorView[1] and will pop-up a color chooser when it is double clicked.

6 Multiple Representation of PDA Data

Students may observe the pushdown automaton through the following four views:

- Tape view
- Stack view
- Path view
- Automata Image view

6.1 Tape View

The tape view provides a visual representation of the string being tested by the PDA. In the tape view, each character of the input string is displayed in order from left to right across the automata display window in a scrollable panel. The tape view panel will grow to accommodate large strings.



When the animation starts, the user is prompted for the input string, which is then added to the tape and displayed in the tape view. The tape view has a small arrow that moves along the bottom indicating the next character to be read. Each time a character is read, the arrow moves

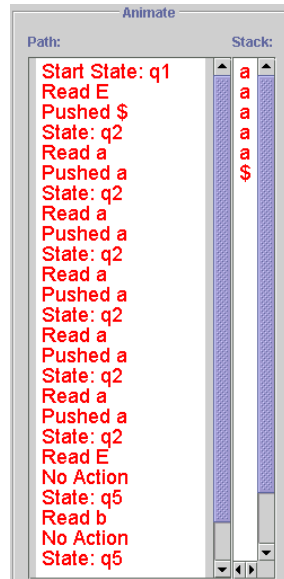
accordingly, however, it does not move when the algorithm moves to a new state on Epsilon. The arrow moves in time with the animation so that the student knows exactly which character is being tested at any moment.

6.2 Stack View

The stack view provides visual feedback from the stack object as the machine processes the tape. Each time a symbol is pushed on or popped off the stack, that symbol is displayed to the student at the top of the stack view. Possible stack actions are:

- Push
- Popped
- No Action

“No Action” indicates that although the algorithm may have read a tape symbol and/or proceeded to the next state, the transition indicated a no-op and therefore the stack remained unchanged.



The stack view will scroll to accommodate a growing stack. When the animation completes, any existing data on the stack will remain in the stack view so the student can see the result of the algorithm.

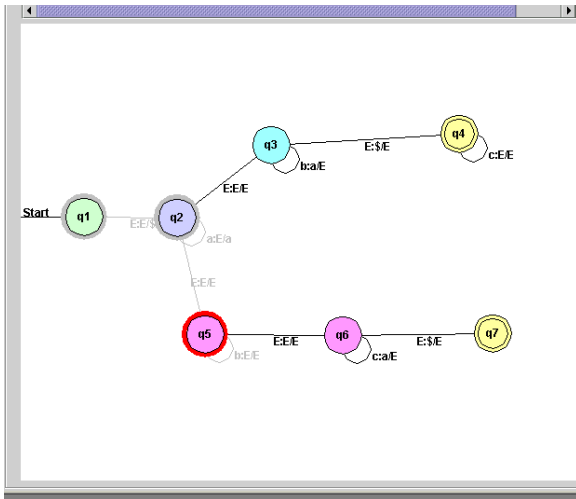
6.3 Path View

The path view displays in English what actions happen to the machine at each step. Every time the algorithm proceeds to a new state, the path view reports the current symbol read from the tape (or Epsilon), the action on the stack, and the new state reached.

The path view is crucial to the students’ understanding of pushdown automata. The path view provides an alternative representation of the stack, tape, and automata image views in easy-to-read English. When the animation completes, the student can use the path view as a transcript of each and every operation performed during the simulation. Students welcome the departure from obscure code-like formal language descriptions when using the IPAA as a learning tool.

6.4 Automata Image View

The automata image view presents the automata to the student the way the student would see it on a chalkboard. The student builds the automaton in this window and when finished, can watch the algorithm traverse the automaton as well.



When the simulation runs, the current state is highlighted by a bright halo. When the algorithm moves to the next state, the color of the halo around the previous state changes, denoting that it was visited. Likewise, the transitions also display different colors indicating the current and visited paths. The next section will explain how the animation colors are chosen and controlled.

7 Algorithmic Animation with Control



The animation controls are located in the bottom left corner. Students can choose the animation speed as well as the colors of active and visited states. There are 3 buttons to start, stop, and step through the animation. Students also have

the option of running the animation in *step-by-step* or *run to completion* mode. In *step-by-step* mode, the animation stops after each step and only performs the next step when the user clicks the *Next* button. The *Next* button is only enabled when running in *step-by-step* mode. In *run to completion* mode, the animation performs a slight pause between each step. When operating in *run to completion* mode, the pause length between each step is controlled by the speed slider.

8 A Starting Point from which to Explore

The IPAA allows users to load and save automata to a file. The ability to load automata from a file allows educators to provide students with a set of pre-constructed automata to experiment with. From there, students may modify the existing automata or create their own.

The Save/Load option in the IPAA is made possible through the use of the Stringable interface in JPT [1]. When an automaton is saved to a file, the model is actually encoded into string form and saved as ASCII text

to a file. When the student later goes to open that automaton, the string is decoded and the model is reconstructed. The model is then used to restore the view state of the program. All visual data, including state color, radius, and position in the window, is stored in the data model. Therefore the automaton is safely preserved when it is saved to a file.

9 The Algorithm

The algorithm behind the automata animation involves searching forward through the automata for a valid path and then backtracking each time a dead end is hit. The algorithm uses two stacks to do this. The first stack acts as the stack for the PDA, and the second stack stores all of the actions performed at each step in the algorithm so that the state may be restored when it comes time to backtrack. Performing the automata traversal entails recursion over six methods. Each of the six methods performs a specific part of the traversal and are as follows:

- Decide whether or not to progress forward.**
 Check the state of the machine and decide whether or not to continue on the current path. If a successful path has been found or a dead end has been reached, return. Otherwise forward the current state and tape position to the next method for processing.
- Locate accessible states**
 Find all states reachable from the current state. For each state found, pass the current state, the reachable target state, tape information, and the transitions between the two states to the next method.
- Evaluate accessible transitions**
 Examine the current state and target state sent from the previous task. Compare the transitions between them with the top of the stack and the current tape position. If the transition can be taken from the current state to the target state, pass all data forward to the next method.
- Progress forward**
 Visit the target state. Call the next method to advance the machine. Call the undo method to enable backtracking.
- Perform stack action and update the machine**
 A successful path node has been found. Add the node to the given path and get the stack action corresponding to the transition in the node. Before performing the action, save its equal and opposite reaction to the undo stack so that the state of the machine can be restored later. Perform the action, move the tape pointer to the next character to be read, and recurse back to the

first method, passing the new state and tape pointer.

- **Undo forward progress**

Pop the undo action off of the undo stack and perform it. This returns the automata stack to its previous state.

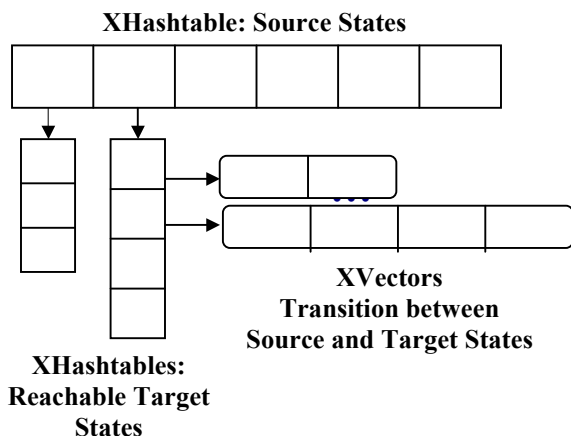
This algorithm searches depth-first for the first successful path. Each time the algorithm reaches a dead end in the automaton, the current path is added to the list of paths found. If the current path is a successful path the algorithm halts the search and returns immediately. Otherwise, it will continue searching, adding each unsuccessful path found to the path list until it can search no more. The algorithm returns the list of paths found and the program chooses a path to animate according to the following rules:

- If a successful path is found, it is animated.
- If no successful paths are found, the longest path is animated.

I made the decision to animate only one path because I felt that watching the real-time traversal would be too confusing for students whose real concern was how to find a successful path through the automata given an input string.

10 The Data Structure

The main role of the data structure is to preserve the relationship between states and transitions in an efficient manner. The automata data model is a three-dimensional structure with vectors inside hash tables inside hash tables. The first hash table contains keys representing all possible source states in the automata. For each key there is a corresponding hash table whose keys are the target states reachable. The second hash table has vectors as values that contain all of the transitions that connect the source state to the target state.



By nesting the collections inside each other, I could preserve the connections between states and eliminate

searching by returning a list of reachable states in one lookup. For example, if I know that my start state is Q1, I can retrieve a list of states that I can reach from Q1 in one lookup. Once I have that list, I can easily get a list of the transitions that connect Q1 with the other state. From there, I simply compare the transition with the tape symbol and top of the stack to determine if it can be traversed.

11 Conclusion and Lessons Learned

Designing the IPAA has been a fabulous experience for me as an undergraduate student in computer science. Not only have I created a tool that I can share with friends learning Automata Theory, but I have also learned a number of lessons about object-oriented design:

First, recursion is not just for algorithms. Nested data structures can be extremely powerful as well as efficient. Students should be exposed to nested collections from the very beginning of CS1.

Second, just because the code compiles and runs does not mean that it is the best implementation. Students should be taught to constantly refactor their code to provide as much encapsulation as possible.

Third, toolkits should be used whenever possible. If a chunk of code appears more than once, it should be encapsulated as a tool. Students should be encouraged to use toolkits (and create their own) whenever possible. The JPT toolkit, in particular, is an excellent toolkit both for pedagogy and GUI design.

These three lessons paired with the blueprint presented in this paper have made the IPAA a useful tool for teaching Automata Theory. Educators are encouraged to use these ideas to create their own visual tools for CS pedagogy. Using visualization tools in the classroom and making them available to the students will motivate students to explore and learn outside the classroom. Students will appreciate the departure from the traditional chalkboard and lecture environment.

References

- [1] Proulx, Viera K., Raab, Jeff, and Rasala, Richard, *Java Power Tools: Model Software for Teaching Object Oriented Design*, SIGSCE Bulletin, 33(1), 2001, 297-301.
- [2] Sipser, Michael, *Introduction to the Theory of Computation*, Brooks-Cole, 1997.
- [3] Proulx, Viera K, Rasala, Richard, and Raab, Jeff, *Java Power Tools: A Foundation for Interactive HCI Exploration*, Systems, Social and Internalization Design Aspects of Human-Computer Interaction: Volume 2 of the Proceedings of HCI International 2001, August 5-10, 2001, New Orleans, LA, Lawrence Eldbaum Associates, 755-759.