Building Your First Pushdown Automaton

Contents

Definition

JFLAP defines a nondeterministic pushdown automaton (NPDA) M as the septuple

$M = (Q, \Sigma, \Gamma, \delta, q_s, Z, F)$ where

Q is a finite set of states $\{q_i \mid i$ is a nonnegative integer$\}$
Σ is the finite input alphabet

Γ is the finite stack alphabet
δ is the transition function, $\delta : Q \times \Sigma^* \times \Gamma^* \rightarrow$ finite subsets of $Q \times \Gamma^*$
q_s (is member of Q) is the initial state

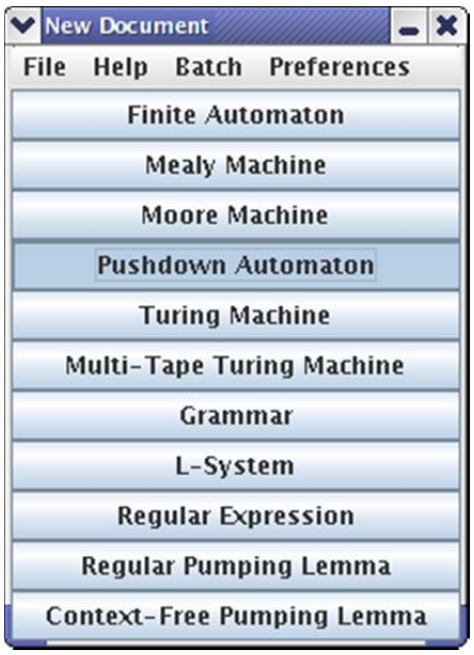
Z is the start stack symbol (must be a capital Z)
F (is a subset of Q) is the set of final states

Note that this definition includes deterministic pushdown automata, which are simply nondeterministic pushdown automata with only one available route to take.

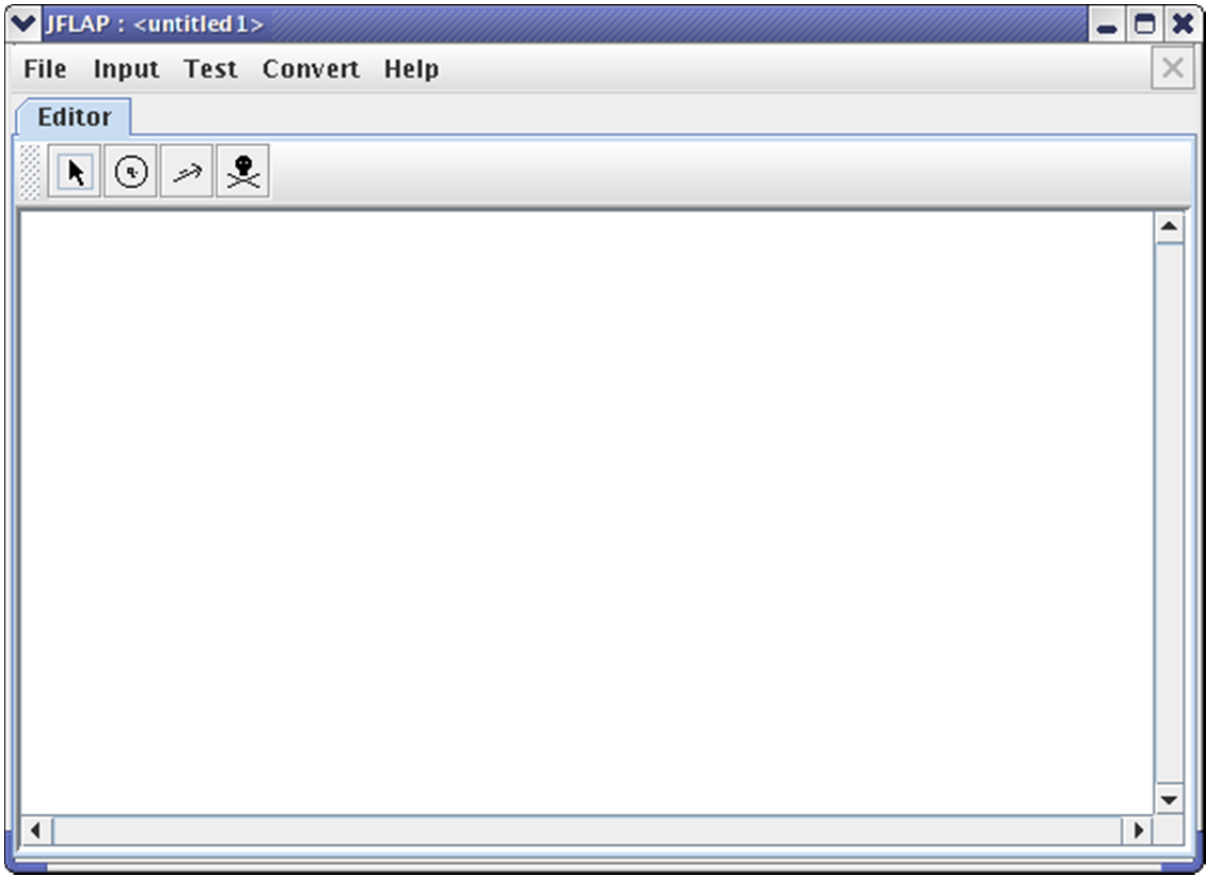
How to Create an Automaton

For knowledge of many of the general tools, menus, and windows used to create an automaton, one should first read the tutorial on finite automata. This tutorial will principally focus on features and options that differentiate pushdown automata from finite automata.

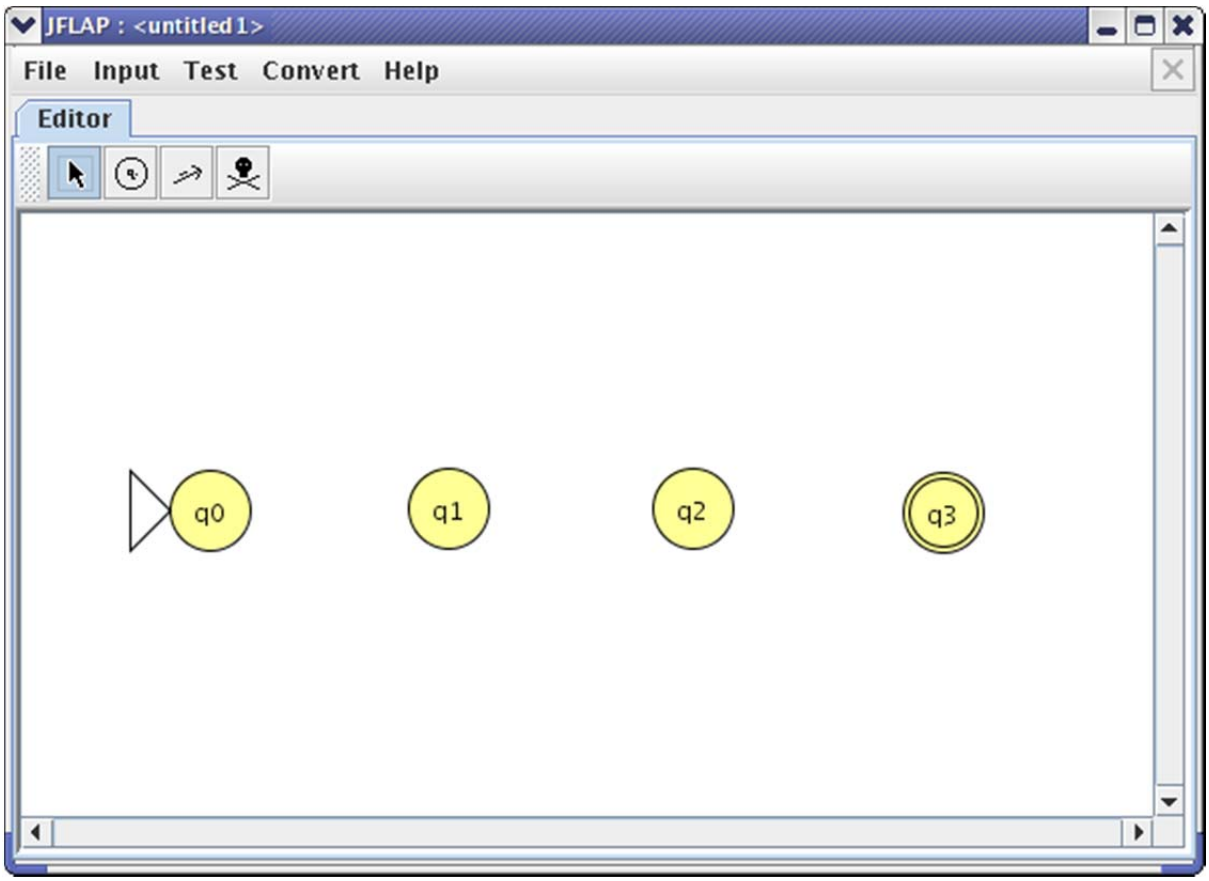
We will begin by constructing a deterministic NPDA for the language $L = \{a^n b^n : n > 0\}$. Our approach is to push an "a" on the stack for each "a" in the input, and to pop an "a" off the stack for each "b" in the input. To start a new NPDA, start JFLAP and click the **Pushdown Automaton** option from the menu, as shown below:



One should eventually see a blank screen that looks like the screen below. There are many of the same buttons, menus, and features present that exist for finite automata. However, there are a few differences, which we will encounter shortly.



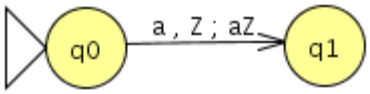
Four states should be enough for the language $L = \{a^n b^n : n > 0\}$. Add four states to the screen, setting the initial state to be q0 and the final state to be q3. The screen should look similar to one below.



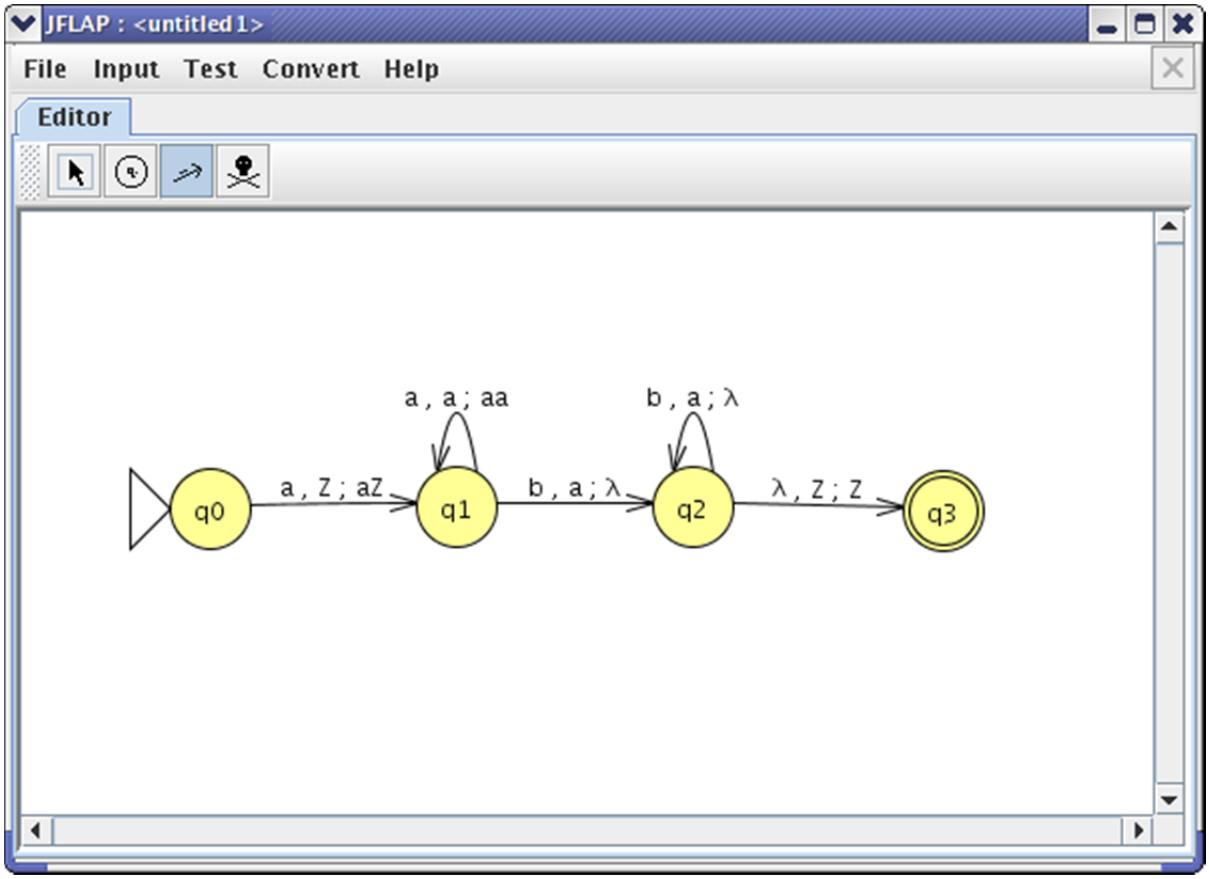
Now its time to add the transitions. Attempt to add a transition between the states q0 and q1. However, there is something different about these transitions in comparison to those created with finite automata. One will notice that there are three inputs instead of one.

The value in the first box represents the input to be processed, the value in the second box represents the current value at the top of the stack, and the final value represents the new value to be pushed onto the top of the stack, after popping the value at the top of the stack off. There is no limit to the size of the values in any of these boxes.

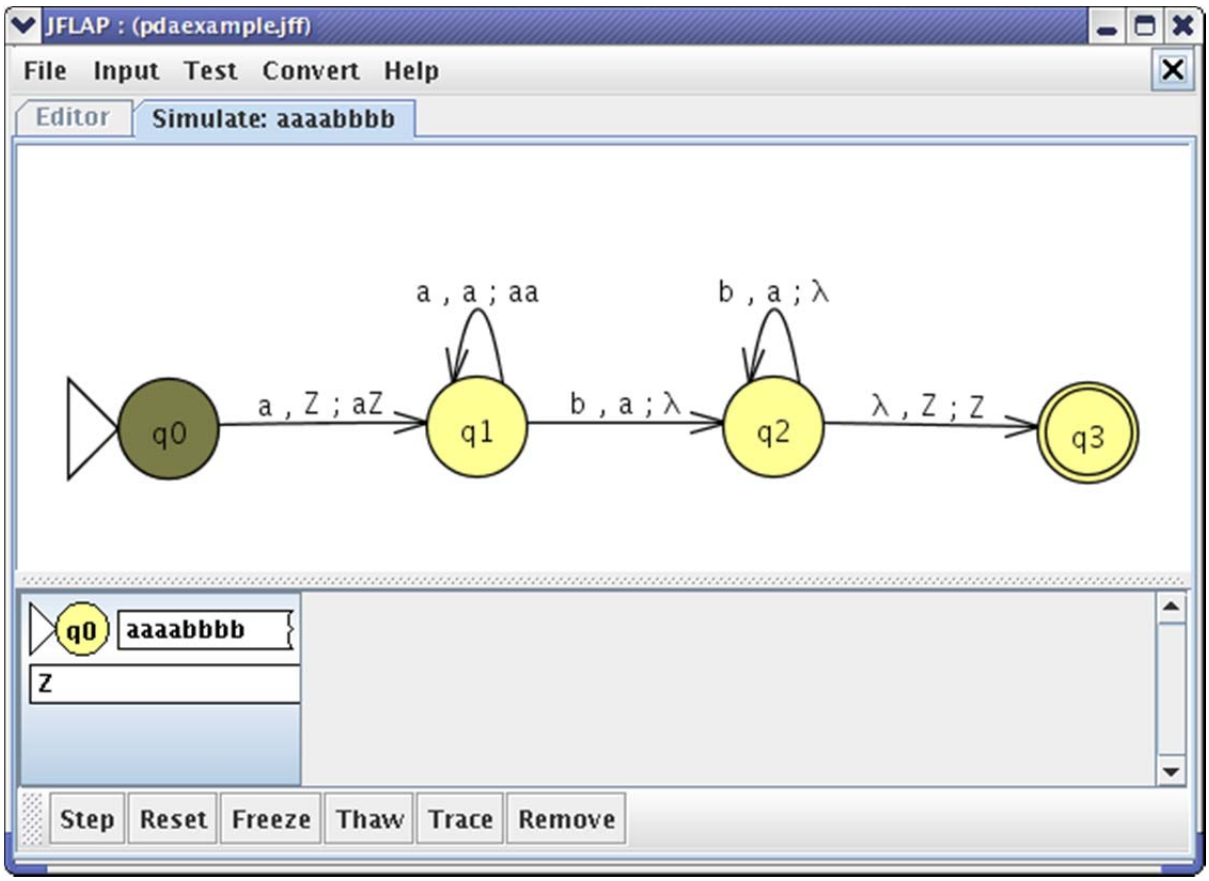
Now, it's time to add input. To change the transition from the default, click on the first box. Enter a value of “a” for the first box, a value of “Z” for the second box (the default character for the bottom element in the stack), and a value of “aZ” in the third box. Use Tab or the mouse to move between the boxes, and press enter or click the mouse on the screen outside the boxes when done. This transition means the the following. If the first symbol of the input is “a”, the first symbol of the stack is “Z”, and the machine is in state “q0”, then pop off “Z” and push “aZ” onto the stack. This transition thus adds an “a” to the stack if utilized. When done, the area between q0 and q1 should resemble the example below.



Let's finish up the transitions. Add a transition (a, a; aa) between q1 and q1 to finish up the a^n segment. Create the transition (b, a; λ) between q1 and q2 and between q2 and q2 to represent the b^nsegment. Finally, (λ , Z; Z) between q2 and q3 will allow an input to arrive at the final state. When finished, the screen should look like this:



For your convenience, the NPDA that we have just made is available in pdaexample.jff. Now, click on the "Input" menu and select "Step with Closure." It will prompt you for input, so enter "aaaabbbb" (representing a^4b^4). After clicking "OK" or pressing enter, the following screen should come up:

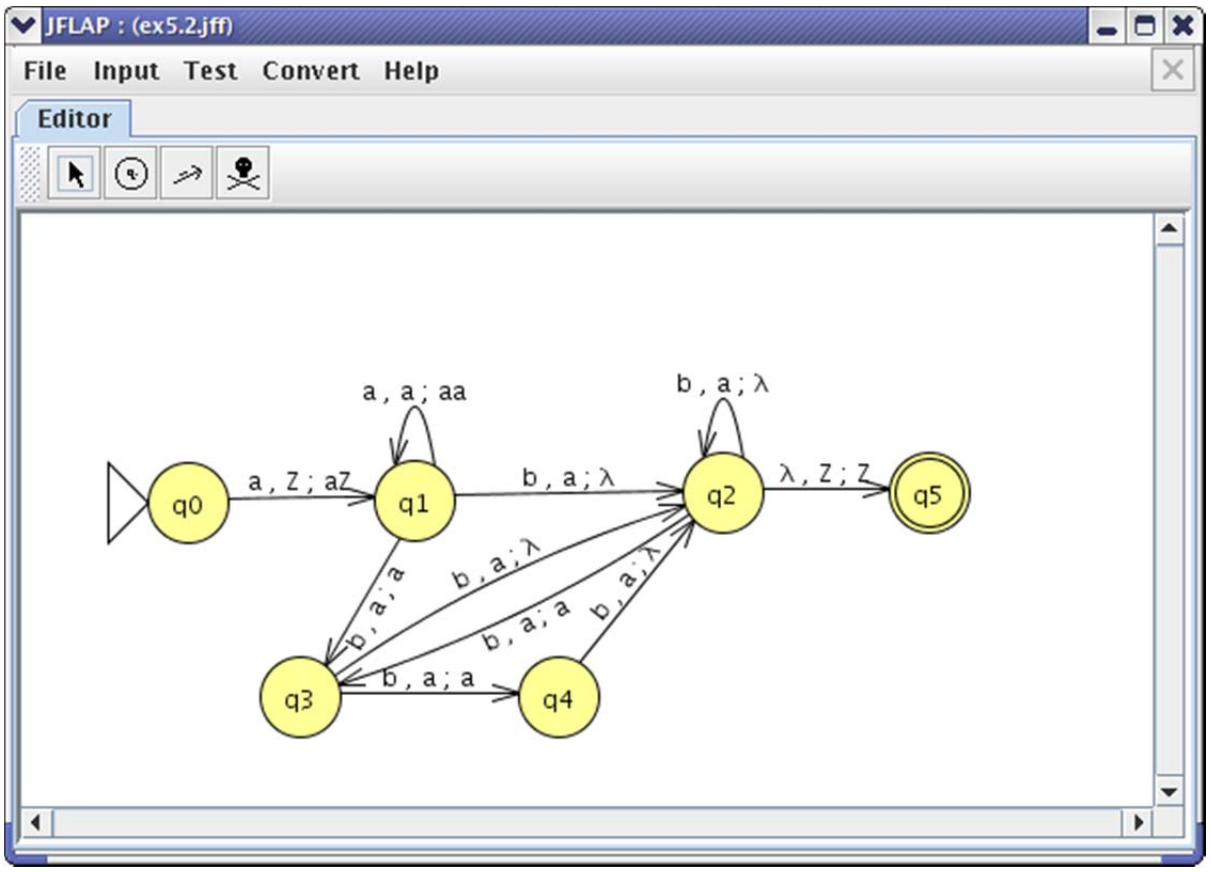


This is very similar to the corresponding screen for finite automata. However, a noted difference is that there is a second box of text in the panel in the lower left. This text box, which currently contains a 'Z', is our stack. If we click the step button, we can see an example of how the stack changes as the simulation runs. There is now be an 'a' in front of the 'Z'. As we continue the simulation, the stack will be at its largest when the a^n values have been processed, but not any b^n values. However, as the *b* values are processed, it will get smaller. The stack will finally end back where it started, with a value of "Z". Thus we finish our first deterministic NPDA.

After the first step **After a^n** **After the first *b* value** **When finished**

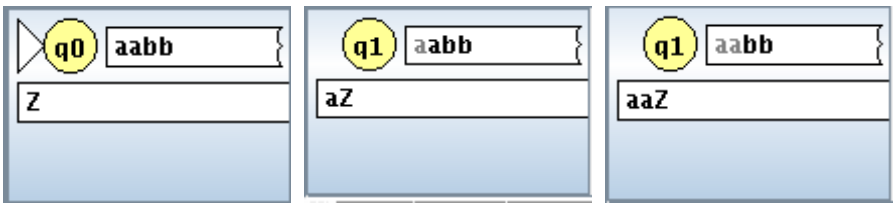
Nondeterministic NPDAs

Nondeterministic NPDAs work in a similar way. One example is the one below, which one can access online here here.



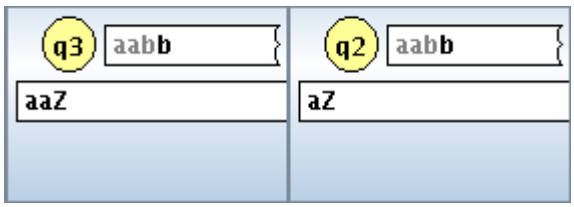
The following is a simulation of "Input → Step by Closure" with the string "aabb":

Start **Step 1** **Step 2**

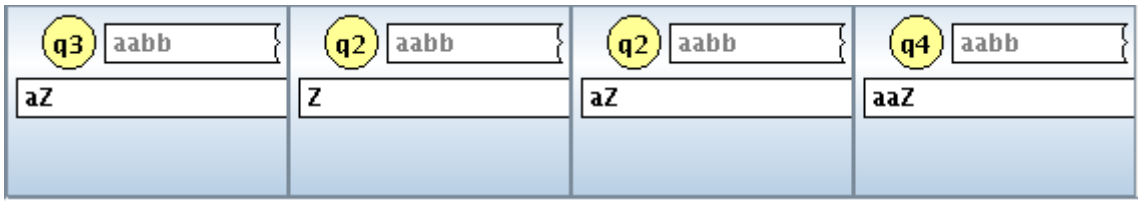


Now is when the nondeterminism begins to assert itself, and the automaton can take more than one path. The last steps cover all possible permutations the program can take, with one path succeeding in the end.

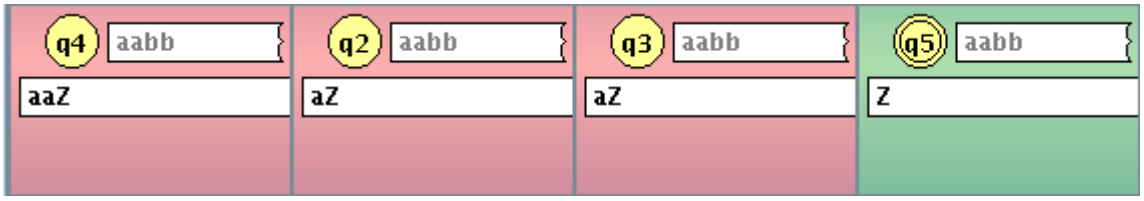
Step 3



Step 4



Results



(Note: This example taken from *JFLAP: An Interactive Formal Languages and Automata Package* by Susan Rodger and Thomas Finley.)