

Building Your First Finite Automaton

Contents

[Definition](#)

[The Editor Window](#)

[Creating States](#)

[Defining Initial and Final States](#)

[Creating Transitions](#)

[Deleting States and Transitions](#)

[Running the Finite Automaton on Multiple Strings](#)

[Building a Nondeterministic Finite Automaton](#)

[Highlighting Nondeterministic States](#)

[Running Input on a Nondeterministic Finite Automaton](#)

- [Producing a Trace](#)
- [Removing Configurations](#)
- [Freezing and Thawing Configurations](#)
- [Resetting the Simulator](#)

[Appendix](#)

- [Adding Notes to JFLAP files](#)
- [Selecting multiple states](#)
- [Layout Algorithms](#)

Definition

JFLAP defines a finite automaton (FA) M as the quintuple $M = (Q, \Sigma, \delta, q_s, F)$ where

Q is a finite set of states $\{q_i \mid i \text{ is a nonnegative integer}\}$

Σ is the finite input alphabet

δ is the transition function, $\delta : D \rightarrow 2^Q$ where D is a finite subset of $Q \times \Sigma^*$

q_s (is member of Q) is the initial state

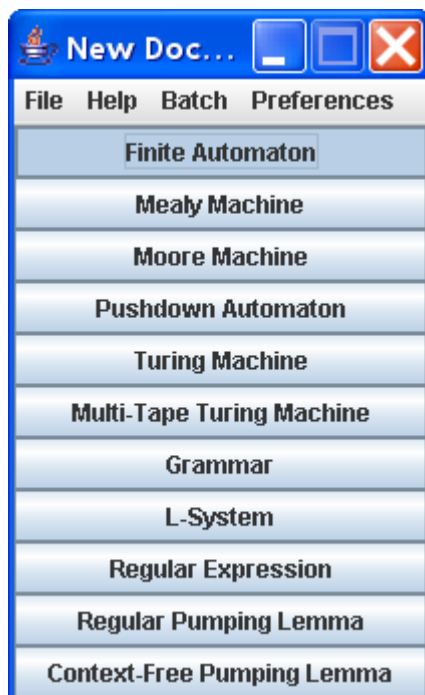
F (is a subset of Q) is the set of final states

Note that this definition includes both deterministic finite automata (DFAs), which we will be discussing shortly, and nondeterministic finite automata (NFAs), which we will touch on later.

Building the different types of automata in JFLAP is fairly similar, so let's start by building a DFA for the language $L = \{a^m b^n : m \geq 0, n > 0, n \text{ is odd}\}$. That is, we will build a DFA that recognizes that language of any number of a 's followed by any odd number of b 's. (Examples taken from *JFLAP: An Interactive Formal Languages and Automata Package* by Susan Rodger and Thomas Finley.)

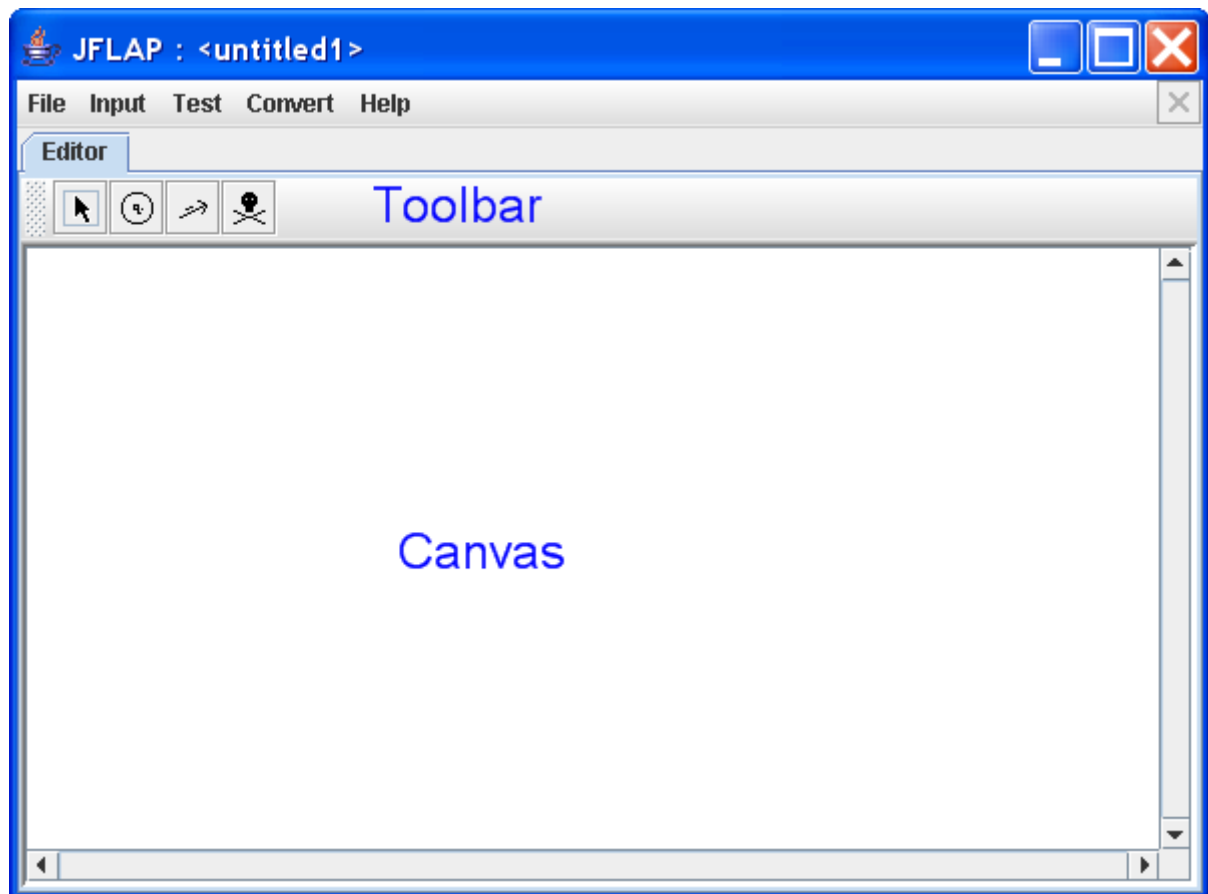
The Editor Window

To start a new FA, start JFLAP and click the **Finite Automaton** option from the menu.



Starting a new FA

This should bring up a new window that allows you to create and edit an FA. The editor is divided into two basic areas: the canvas, which you can construct your automaton on, and the toolbar, which holds the tools you need to construct your automaton.






The editor window

Let's take a closer look at the toolbar.




The FA toolbar

As you can see, the toolbar holds four tools:

- Attribute Editor tool  : [sets initial and final states](#)
- State Creator tool  : [creates states](#)
- Transition Creator tool  : [creates transitions](#)


- Deletor tool : [deletes states and transitions](#)

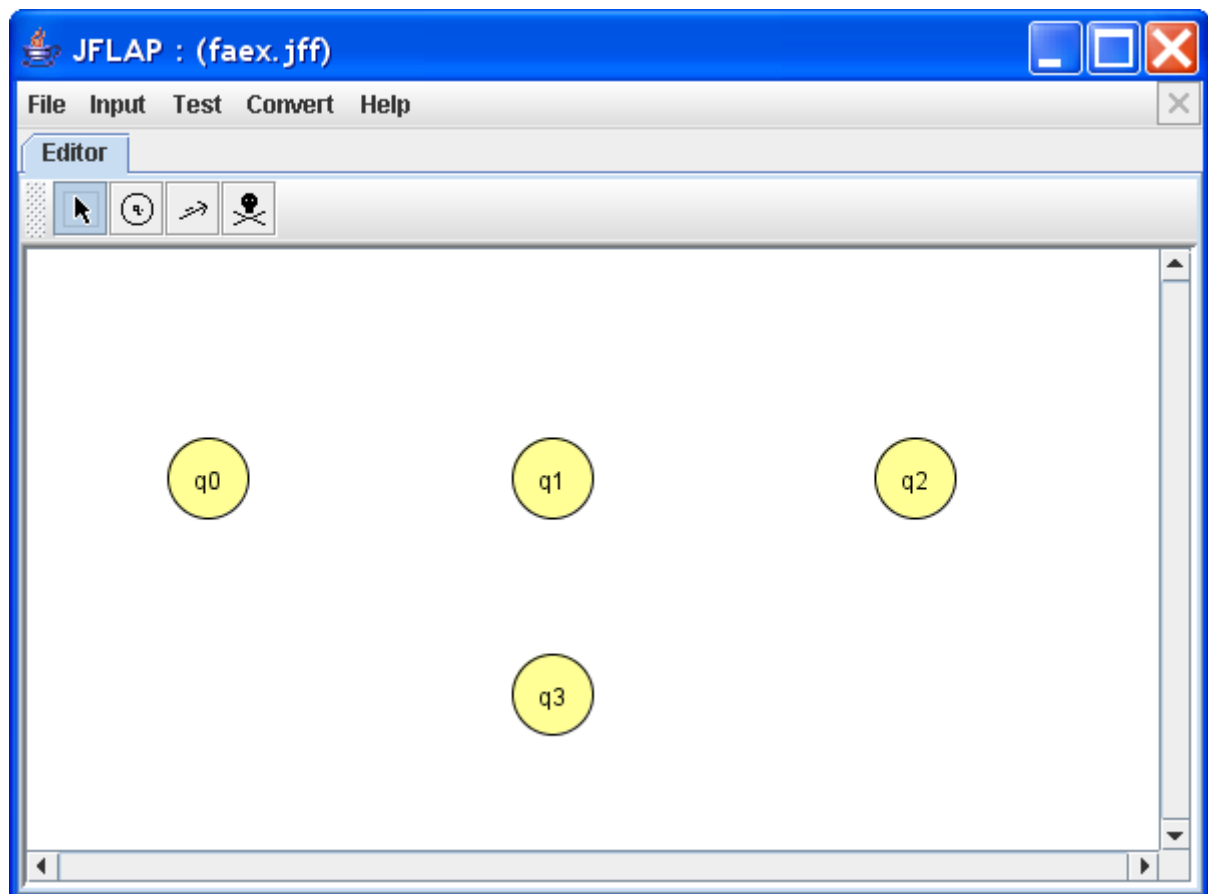
To select a tool, click on the corresponding icon with your mouse. When a tool is selected, it is shaded, as the Attribute Editor tool  is above. Selecting the tool puts you in the corresponding mode. For instance, with the toolbar above, we are now in the Attribute Editor mode.

The different modes dictate the way mouse clicks affect the machine. For example, if we are in the State Creator mode, clicking on the canvas will create new states. These modes will be described in more detail shortly.

Now let's start creating our FA.

Creating States


First, let's create several states. To do so we need to activate that State Creator tool by clicking the  button on the [toolbar](#). Next, click on the canvas in different locations to create states. We are not very sure how many states we will need, so we created four states. Your editor window should look something like this:



States created

Now that we have created our states, let's define initial and final state.

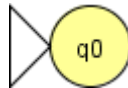
Defining Initial and Final States

Arbitrarily, we decide that q_0 will be our initial state. To define it to be our initial state, first select the Attribute Editor tool  on the [toolbar](#). Now that we are in Attribute Editor mode, right-click on q_0 . This should give us a pop-up menu that looks like this:



The state menu

From the pop-up menu, select the checkbox **Initial**. A white arrowhead appears to the left of q_0 to indicate that it is the initial state.



q_0 defined as initial state

Next, let's create a final state. Arbitrarily, we select q_1 as our final state. To define it as the final state, right-click on the state and click the checkbox **Final**. It will have a double outline, indicating that it is the final state.



q_1 defined as final state

Now that we have defined initial and final states, let's move on to creating transitions.

Creating Transitions

We know strings in our [language](#) can start with a 's, so, the initial state must have an outgoing transition on a . We also know that it can start with any number of a 's, which means that the FA should be in the same state after processing input of any number of a 's. Thus, the outgoing transition on a from q_0 loops back to itself.

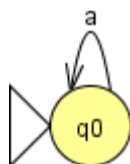
To create such a transition, first select the Transition Creator tool  from the [toolbar](#). Next, click on q_0 on the canvas. A text box should appear over the state:




Creating a transition

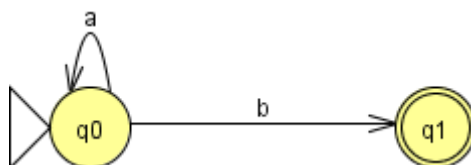
Note that λ , representing the empty string, is initially filled in for you. If you prefer ϵ representing the empty string, select **Preferences : Preferences** in the [main menu](#) to change the symbol representing the empty string.

Type "a" in the text box and press **Enter**. If the text box isn't selected, press **Tab** to select it, then enter "a". When you are done, it should look like this:



Transition created

Next, we know that strings in our [language](#) must end with a odd number of b 's. Thus, we know that the outgoing transtion on b from q_0 must be to a final state, as a string ending with one b should be accepted. To create a transition from our initial state q_0 to our final state q_1 , first ensure that the Transition Creator tool  is selected on the [toolbar](#). Next, click and hold on q_0 , and drag the mouse to q_1 and release the mouse button. Enter "b" in the textbox the same way you entered "a" for the previous transition. The transition between two states should look like this:

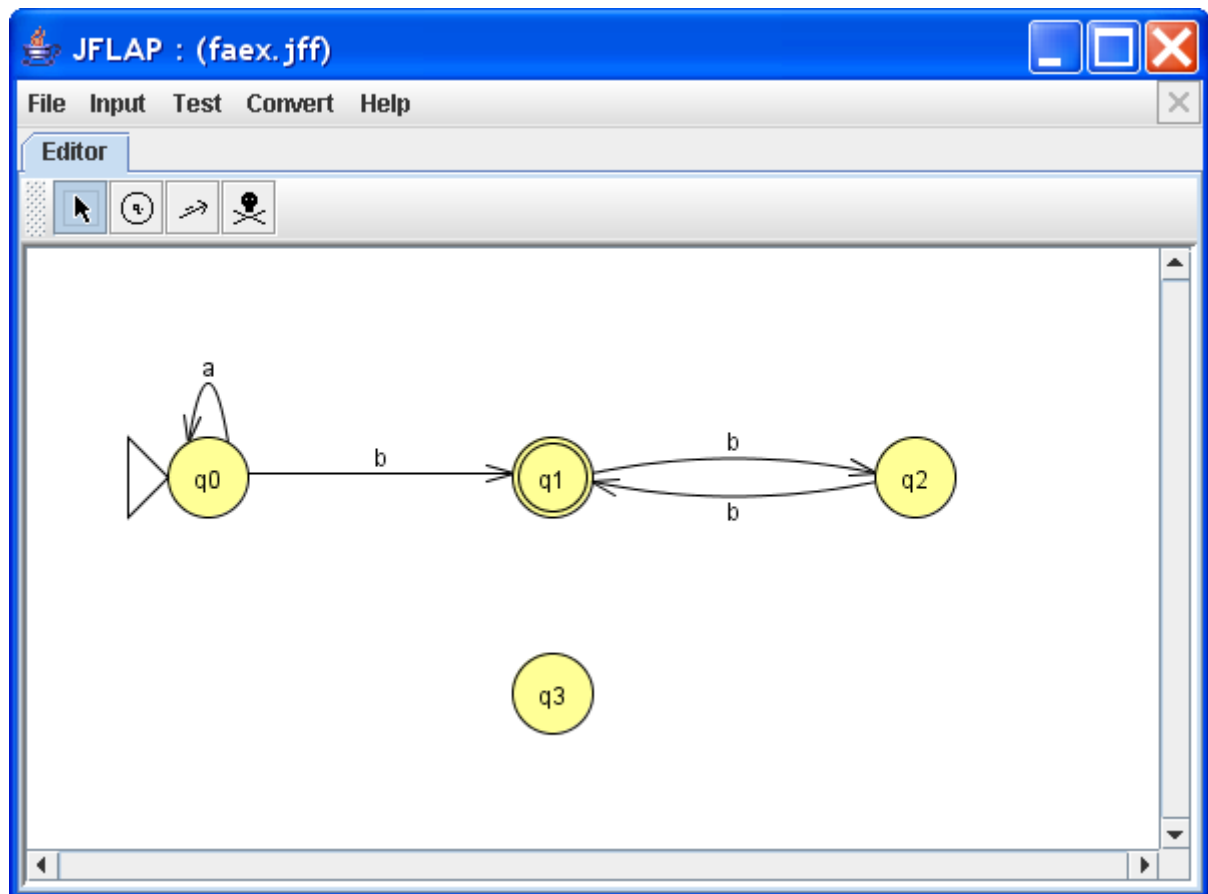


Second transition created

Lastly, we know that only strings that end with an odd number of b 's should be accepted. Thus, we know that q_1 has an outgoing transition on b , that it cannot loop back to q_1 . There are two options for the transtion: it can either go to the initial state q_0 , or to a brand new state, say, q_2 .

If the transition on b was to the initial state q_0 , strings would not have to be of the form $a^m b^n$; strings such as $ababab$ would also be accepted. Thus, the transition cannot be to q_0 , and it must be to q_2 .


Create a transition on b from q_1 to q_2 . As the FA should accept strings that end with an odd number of b 's, create another transition on b from q_2 to q_1 . Your FA is now a full, working FA! It should look something like this:

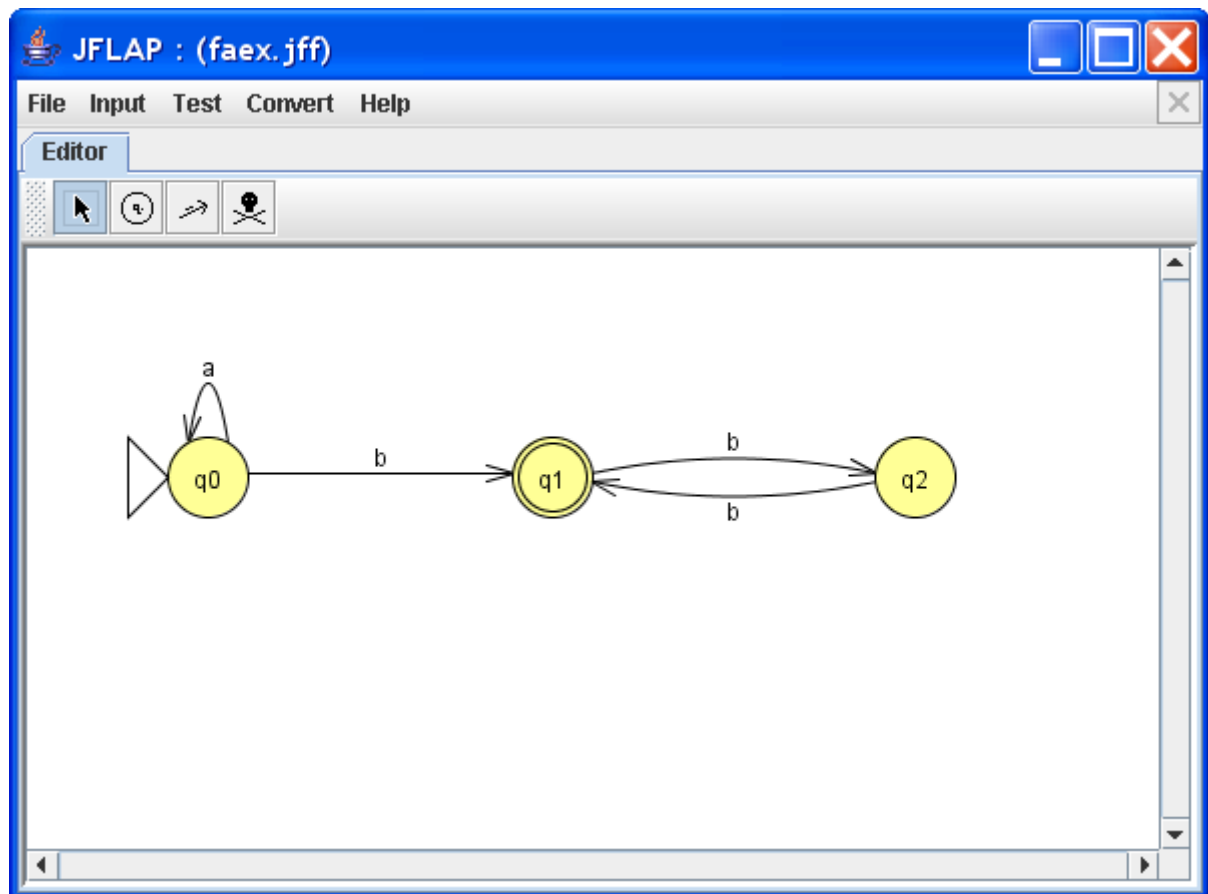


The working FA

You might notice that the q_3 is not used and can be deleted. Next, we will describe how to delete states and transitions.

Deleting States and Transitions

To delete q_3 , first select the Deletor tool  on the [toolbar](#). Next, click on the state q_3 . Your editor window should now look something like this:



q_3 deleted

Similarly, to delete a transition, simply click on the input symbol of the transition when in Deletor mode.

Your FA, [faex.jff](#), is now complete.

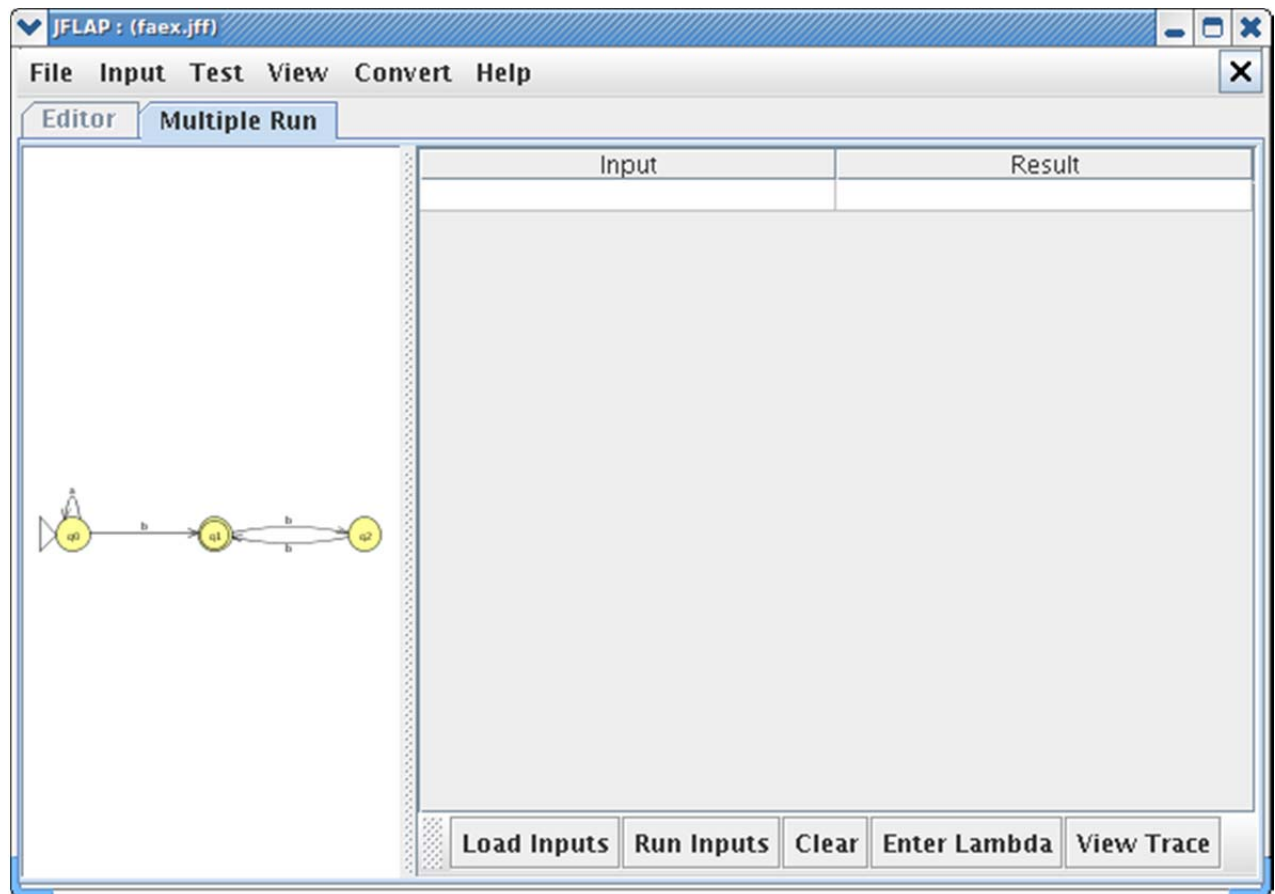
Running the FA on Multiple Strings

Now that you've completed your FA, you might want to test it to see if it really accepts strings from the [language](#). To do so, select **Input : Multiple Run** from the menu bar.



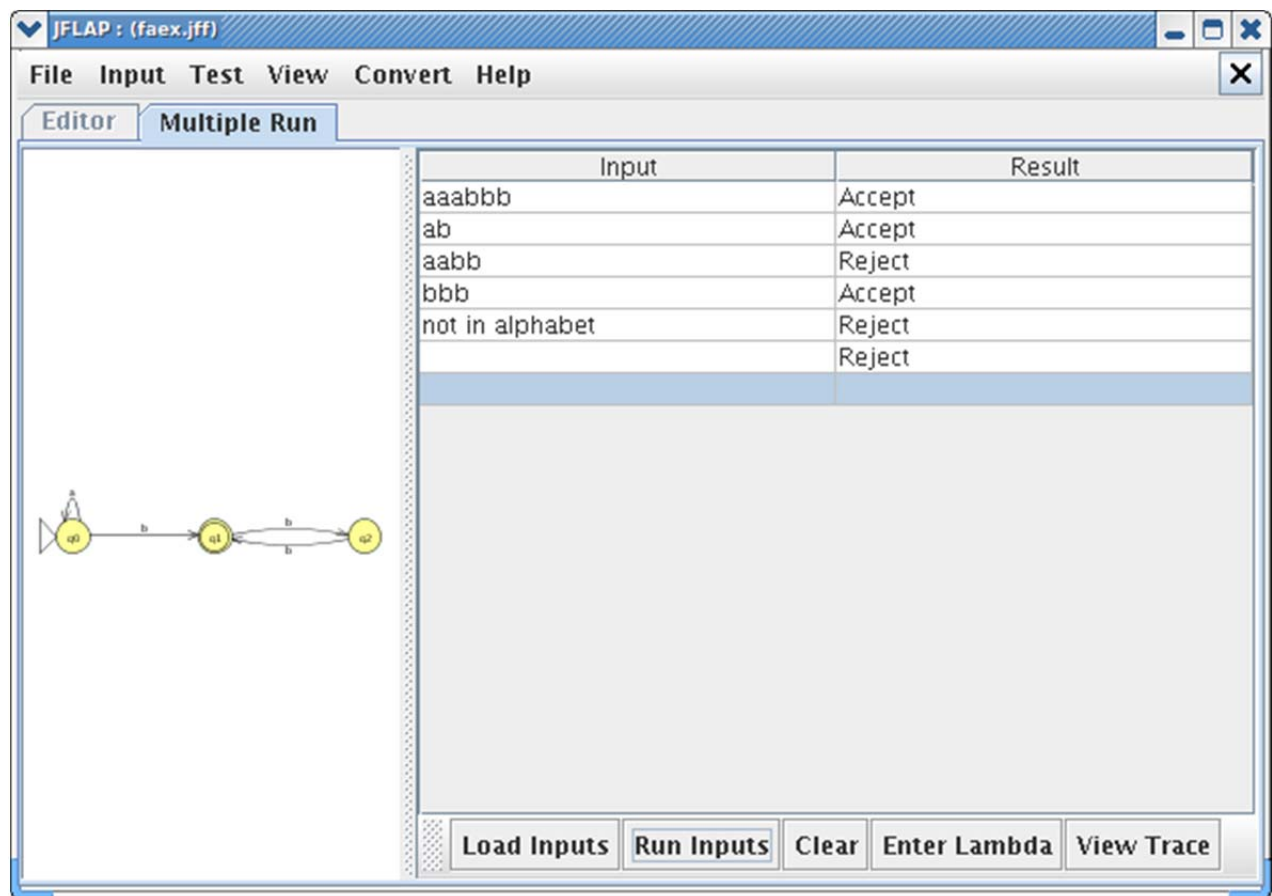
Starting a multiple run tab

A new tab will appear displaying the automaton on the left pane, and an input table on the right:



A new multiple run tab

To enter the input strings, click on the first row in the **Input** column and type in the string. Press **Enter** to continue to the next input string. When you are done, click **Run Inputs** to test your FA on all the input strings. The results, **Accept** or **Reject** are displayed in the **Result** column. You can also load the inputs from file delimited by white space. Simply click on **Load Inputs** and load the file to add additional input strings into multi-run pane.



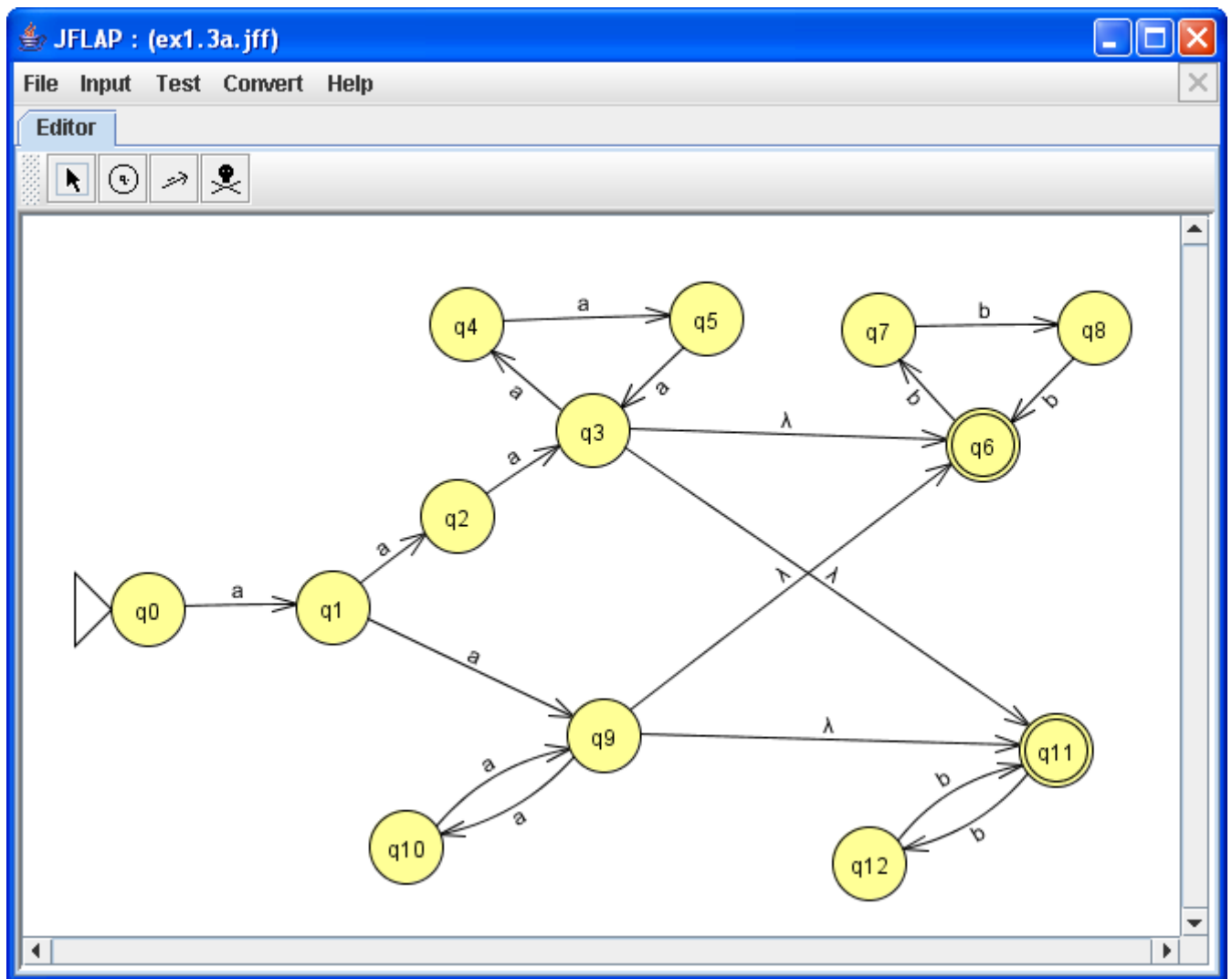
Running multiple inputs

Clicking **Clear** deletes all the input strings, while **Enter Lambda** enters the empty string at the cursor. **View Trace** brings up a separate window that shows the trace of the selected input. To return to the Editor window, select **File : Dismiss Tab** from the menu bar.

Building a Nondeterministic Finite Automaton

Building a nondeterministic finite automaton (NFA) is very much like building a DFA. However, an NFA is different from a DFA in that it satisfies one of two conditions. Firstly, if the FA has two transitions from the same state that read the same symbol, the FA is considered an NFA. Secondly, if the FA has any transitions that read the empty string for input, it is also considered an NFA.

Let's take a look at this NFA, which can be accessed through [ex1.3a.jff](#): (Note: This example taken from *JFLAP: An Interactive Formal Languages and Automata Package* by Susan Rodger and Thomas Finley.)

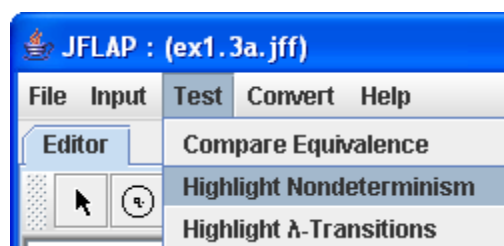


An NFA

We can immediately tell that this is an NFA because of the four λ -transitions coming from q_3 and q_9 , but we might not be sure if we have spotted all the nondeterministic states. JFLAP can help with that.

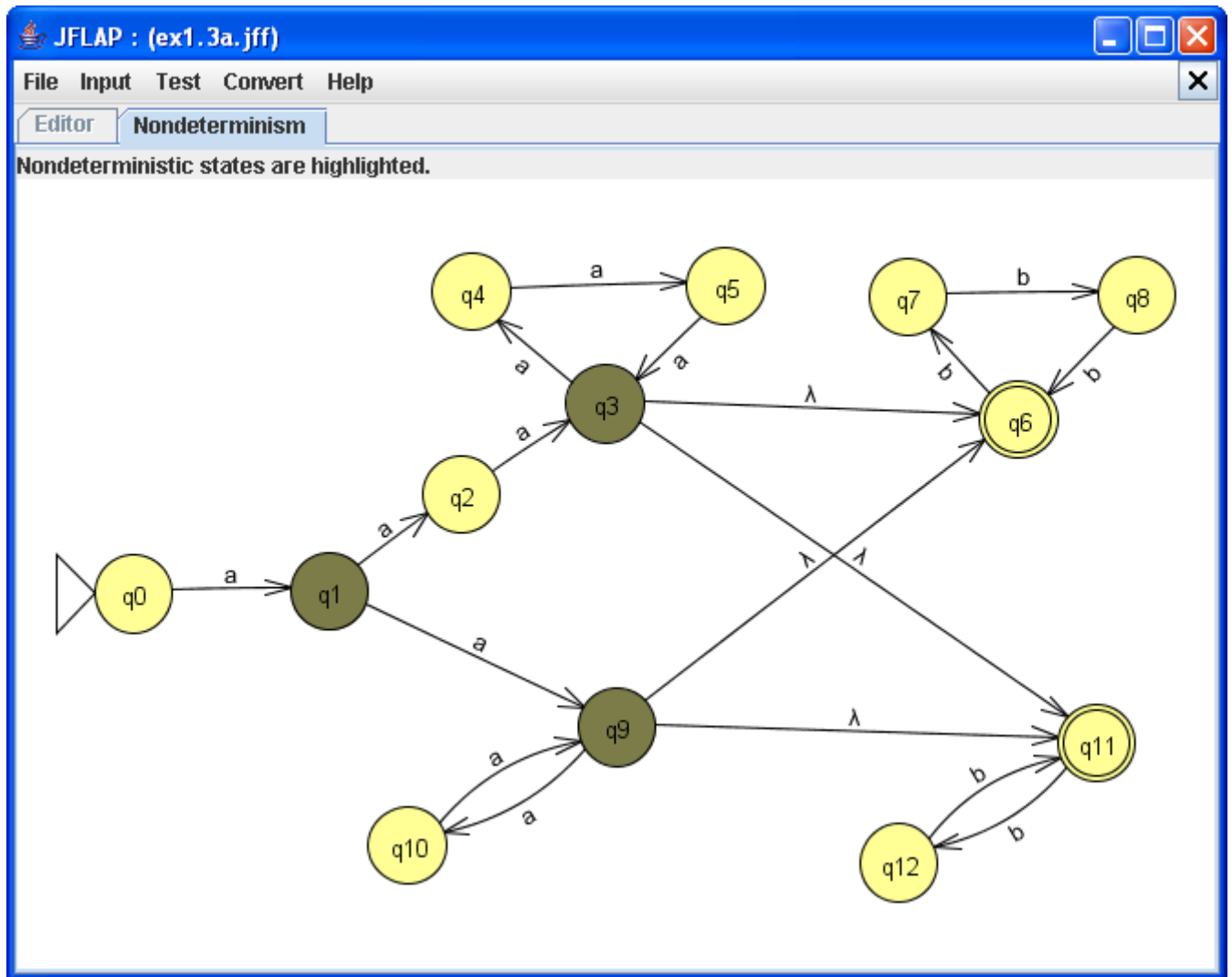
Highlighting Nondeterministic States

To see all the nondeterministic states in the NFA, select **Test : Highlight Nondeterminism** from the menu bar:



Highlighting nondeterministic states

A new tab will appear with the nondeterministic states shaded a darker color:



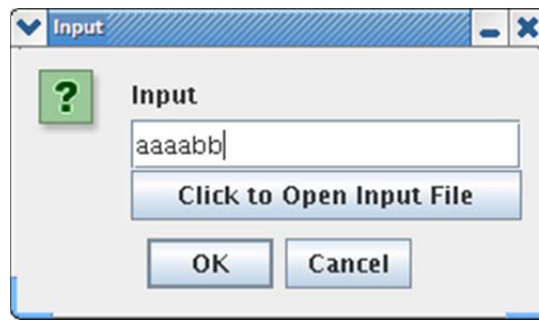
Nondeterministic states highlighted

As we can see, q_3 and q_9 are indeed nondeterministic because of their outgoing λ -transitions. Note that they would both be nondeterministic even if they each had one λ -transition instead of two: only one λ -transition is needed to make a state nondeterministic. We also see that q_1 is nondeterministic because two of its outgoing transitions are on the same symbol, a . Next, we will go through JFLAP's tools for running input on an NFA.

Running Input on an NFA

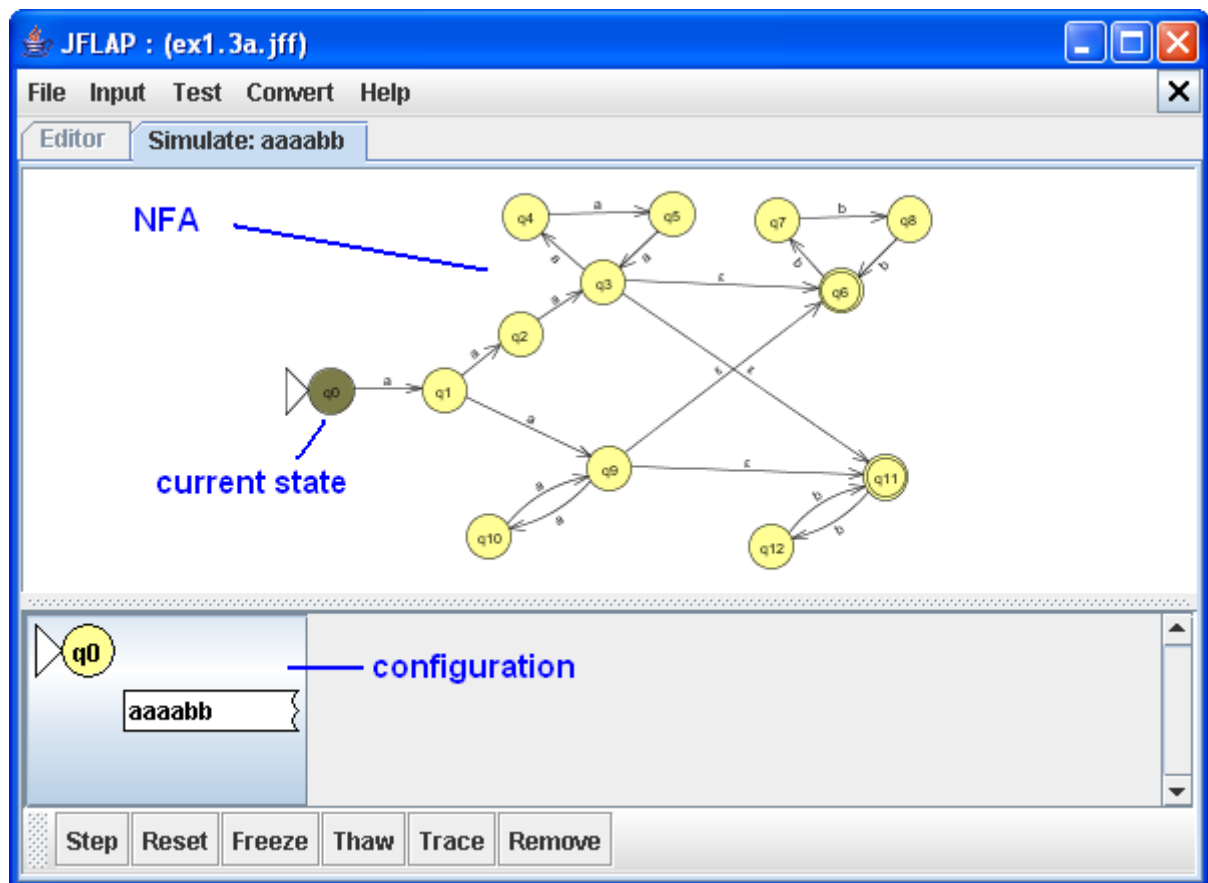
To step through input on an NFA, select **Input : Step with Closure...** from the menu bar. A dialog box prompting you for input will appear. Ordinarily, you would enter the input you wish to step through here. For now, type "aaaabb" in the dialog box and press **Enter**. You can also load the input file instead of typing the string.

NOTE : When loading input from the file, JFLAP determines end of input string by the white space. Thus if there is string "ab cd" in a file, only "ab" will be considered as an input ("cd" will be ignored since there is a white space before them).



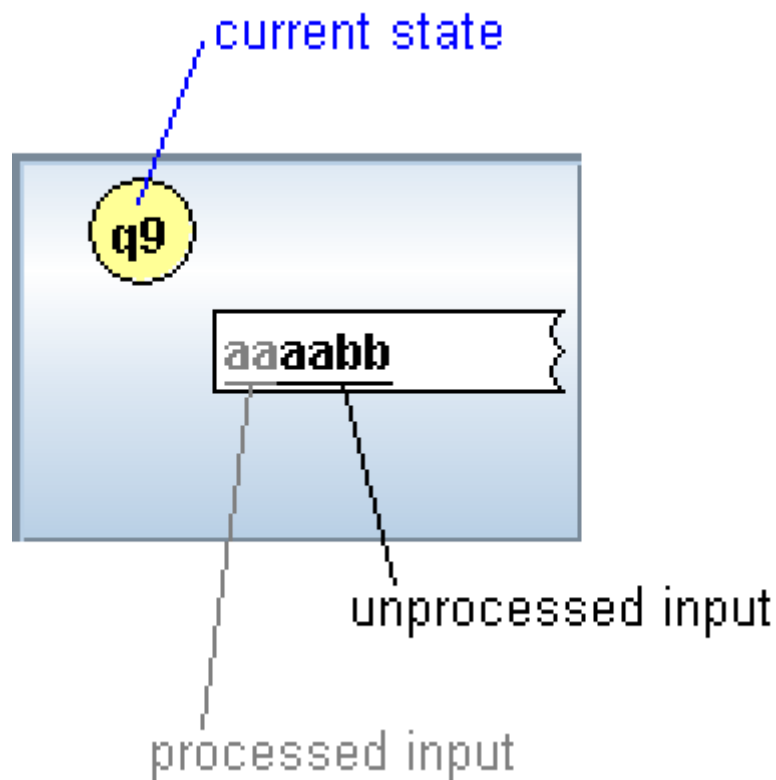
Entering input

A new tab will appear displaying the automaton at the top of the window, and configurations at the bottom. The current state is shaded.



A new input tab

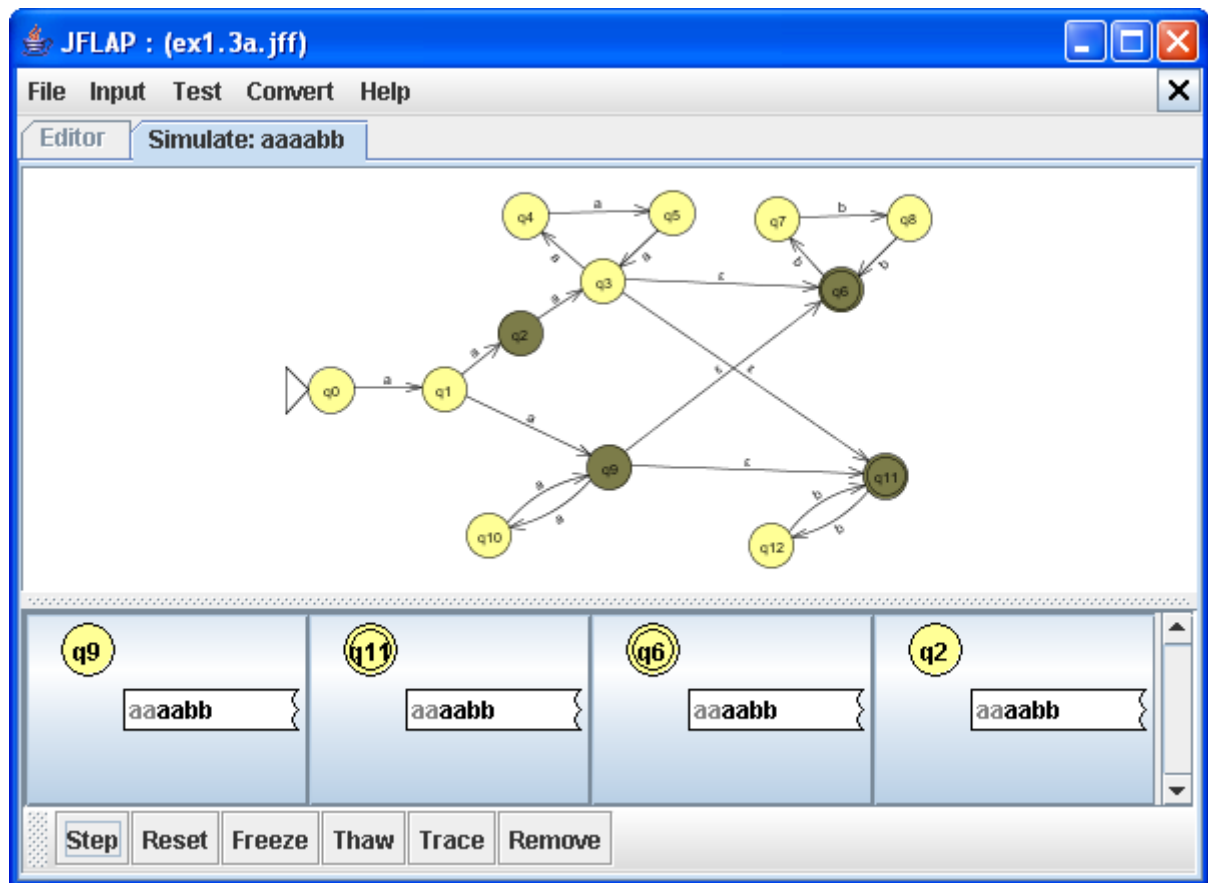
First, let's take a closer look at a configuration:



A configuration icon

The configuration icon shows the current state of the configuration in the top left hand corner, and input on the white tape below. The processed input is displayed in gray, and the unprocessed input is black.

Click **Step** to process the next symbol of input. You will notice q_1 becomes the shaded state in the NFA, and that the configuration icon changes, reflecting the fact that the first a has been processed. Click **Step** again to process the next a . You will find that four states are shaded instead of one, and there are four configurations instead of one.



aa processed

This is because the machine is nondeterministic. From q_1 , the NFA took both a transitions to q_2 and q_9 . As q_9 has two λ -transitions (which do not need input), the NFA further produced two more configurations by taking those transitions. Thus, the simulator now has four configurations. Click **Step** again to process the next input symbol.

first a . After processing the second a , it was in q_{11} . Although q_{11} is not adjacent to q_1 , it can be reached by taking a λ -transition from q_9 . As the simulator tried to process the next a on this configuration, it realized that there are no outgoing a transitions from q_{11} and thus rejected the configuration.

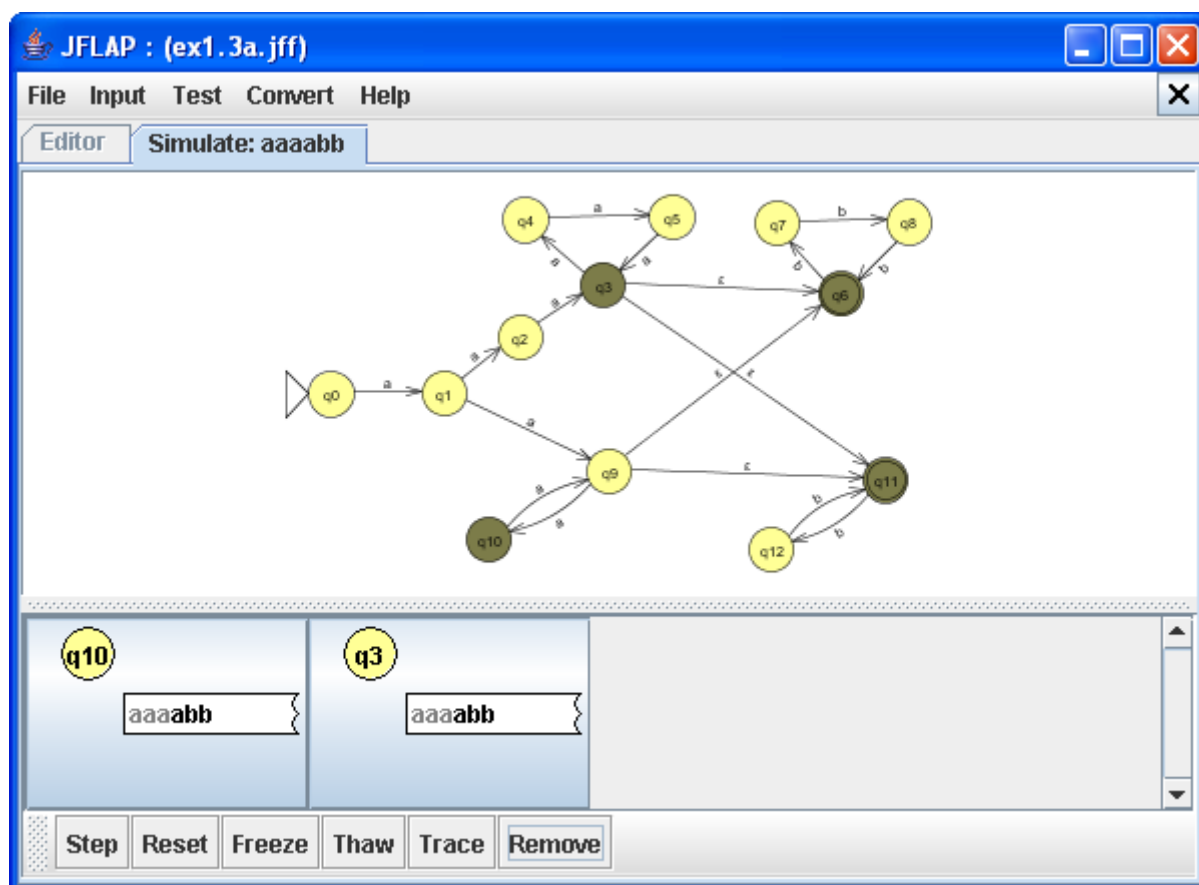
Although rejected configurations will remove themselves in the next step, we can also remove configurations that have not been rejected.

Removing Configurations

Looking at the tracebacks of the rejected configurations, we can tell that any configurations that are in q_{11} or q_6 and whose next input symbol is a will be rejected.

As the next input symbol is a , we can tell that the configurations that are currently in q_6 and q_{11} will be rejected. Click once on each of the four configurations to select them, then click **Remove**. The simulator will no longer step these configurations. (Although we are only removing configurations that are about to be rejected, we can remove any configurations for any purpose, and the simulator will stop stepping through input on those configurations.)

Your simulator should now look something like this:



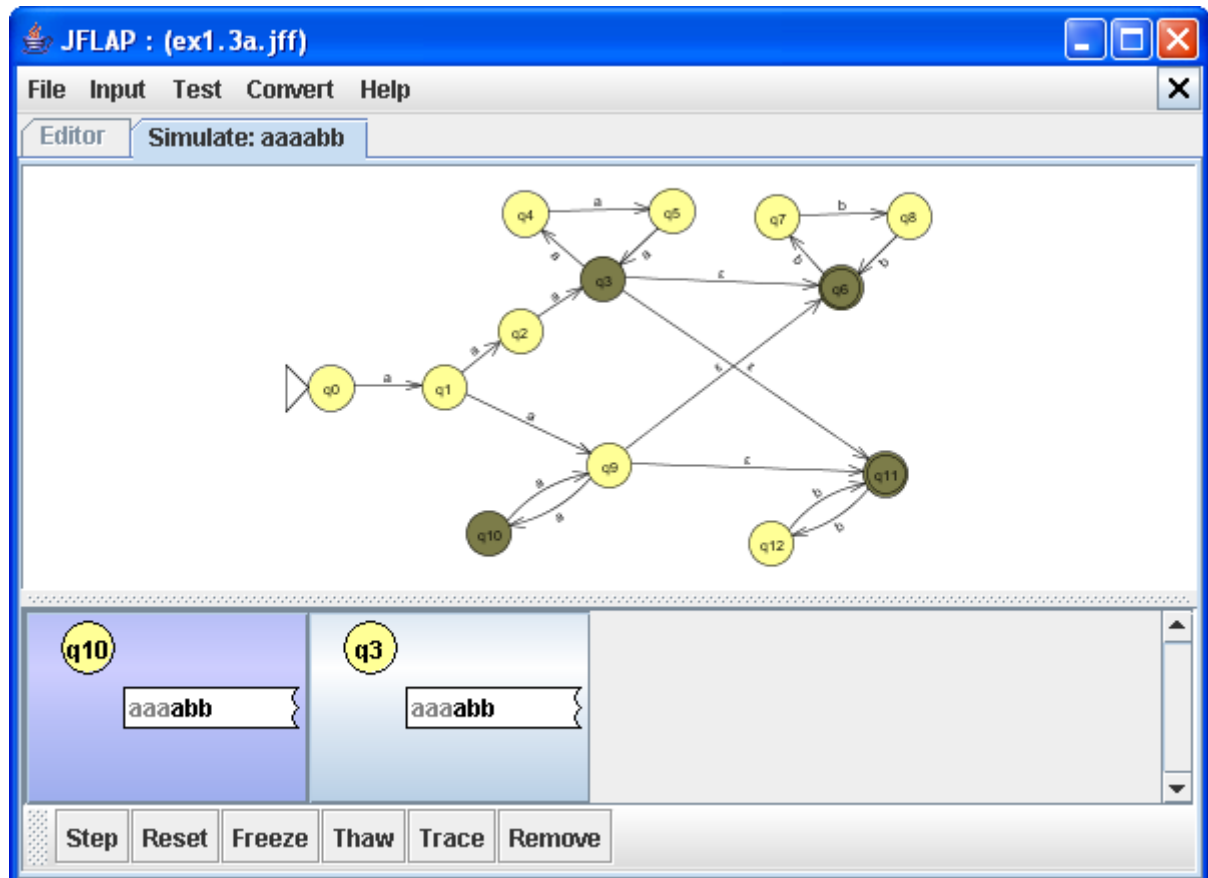
Rejected configurations removed

Now when we step the simulator, the two configurations will be stepped through.

Looking at the two configurations above, we might realize that the configuration on q_3 will not lead to an accepting configuration. We can test our idea out by freezing the other configuration.

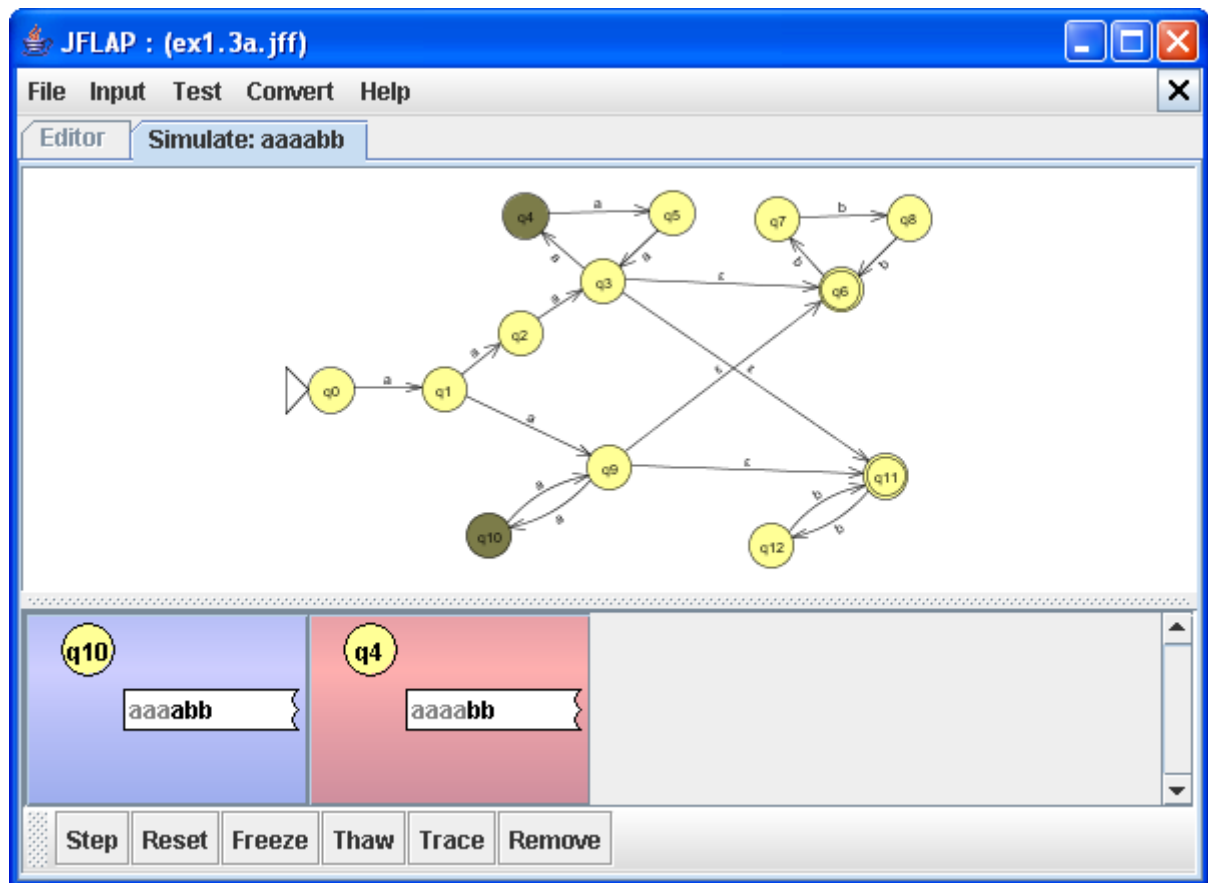
Freezing and Thawing configurations

To freeze the configuration on q_{10} , click on q_{10} once, then click the **Freeze** button. When a configuration is frozen, it will be tinted a darker shade of purple:



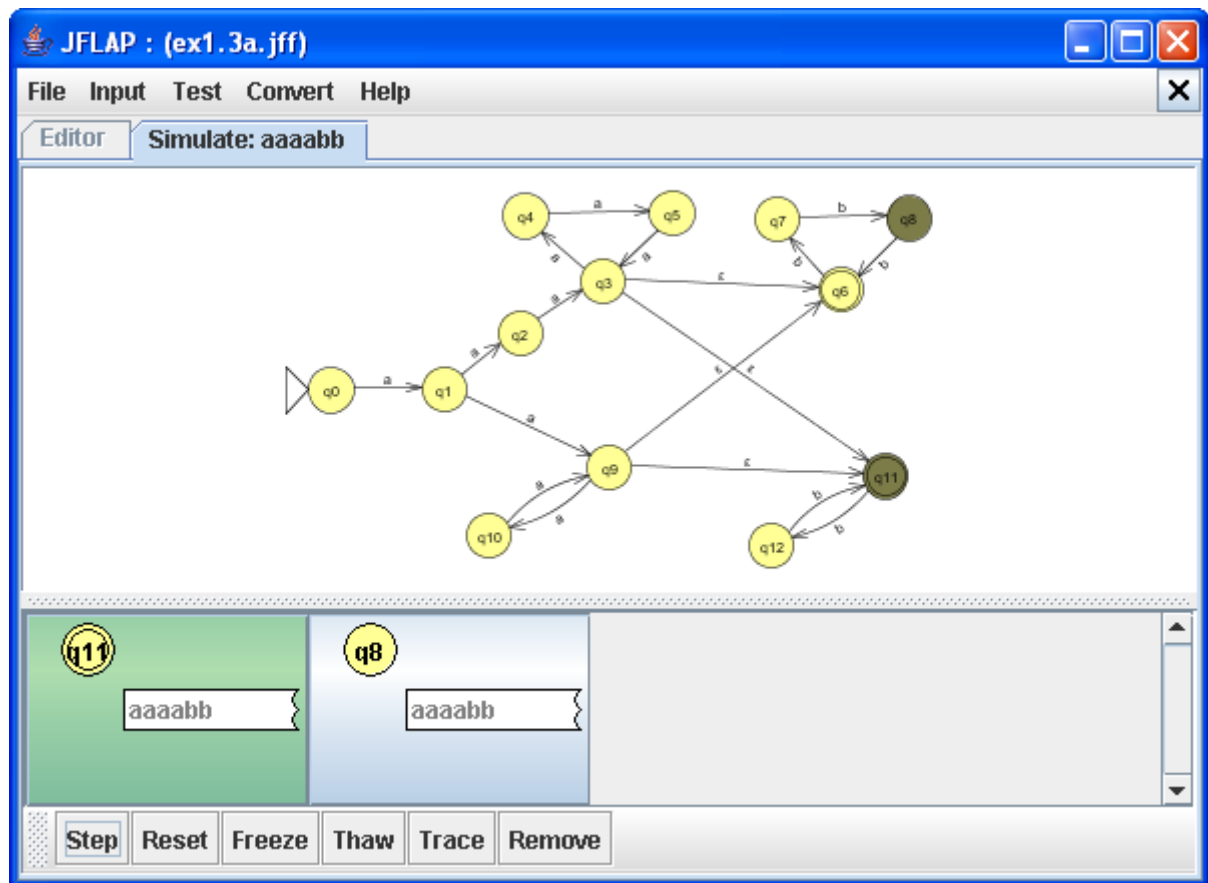
Configuration on q_{10} frozen

With that configuration frozen, as you click **Step** to step through the configuration on q_3 , the frozen configuration remains the same. Clicking **Step** two more times will reveal that the configuration on q_3 is not accepted either. Your simulator will now look like this:



Other configurations rejected

To proceed with the frozen configuration, select it and click **Thaw**. The simulator will now step through input as usual. Click **Step** another three times to find an accepting configuration. An accepting configuration is colored green:



Accepting configuration found

If we click **Step** again, we will see that the last configuration is rejected. Thus, there is only one accepting configuration. However, we might be unsure that this is really the case, as we had removed some configurations. We can double-check by resetting the simulator.

Resetting the simulator

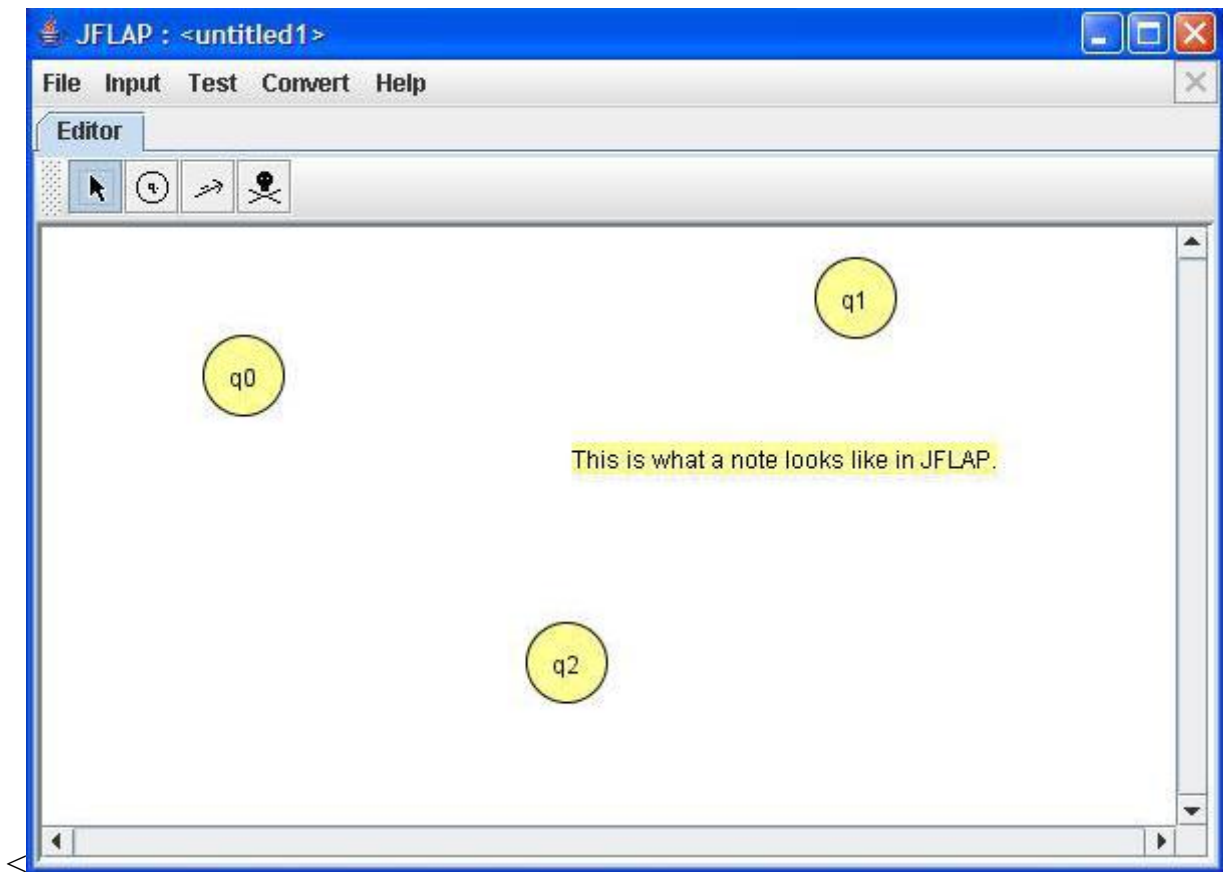
At any point in the simulation, we can restart the entire simulation process by clicking **Reset**. This will clear all the current configurations and restart the simulation. If we click **Reset** and step all the configurations, we will find that there is, indeed, only one accepting configuration.

This concludes the walkthrough, although there is an appendix noting a few more features that JFLAP supports.

Appendix

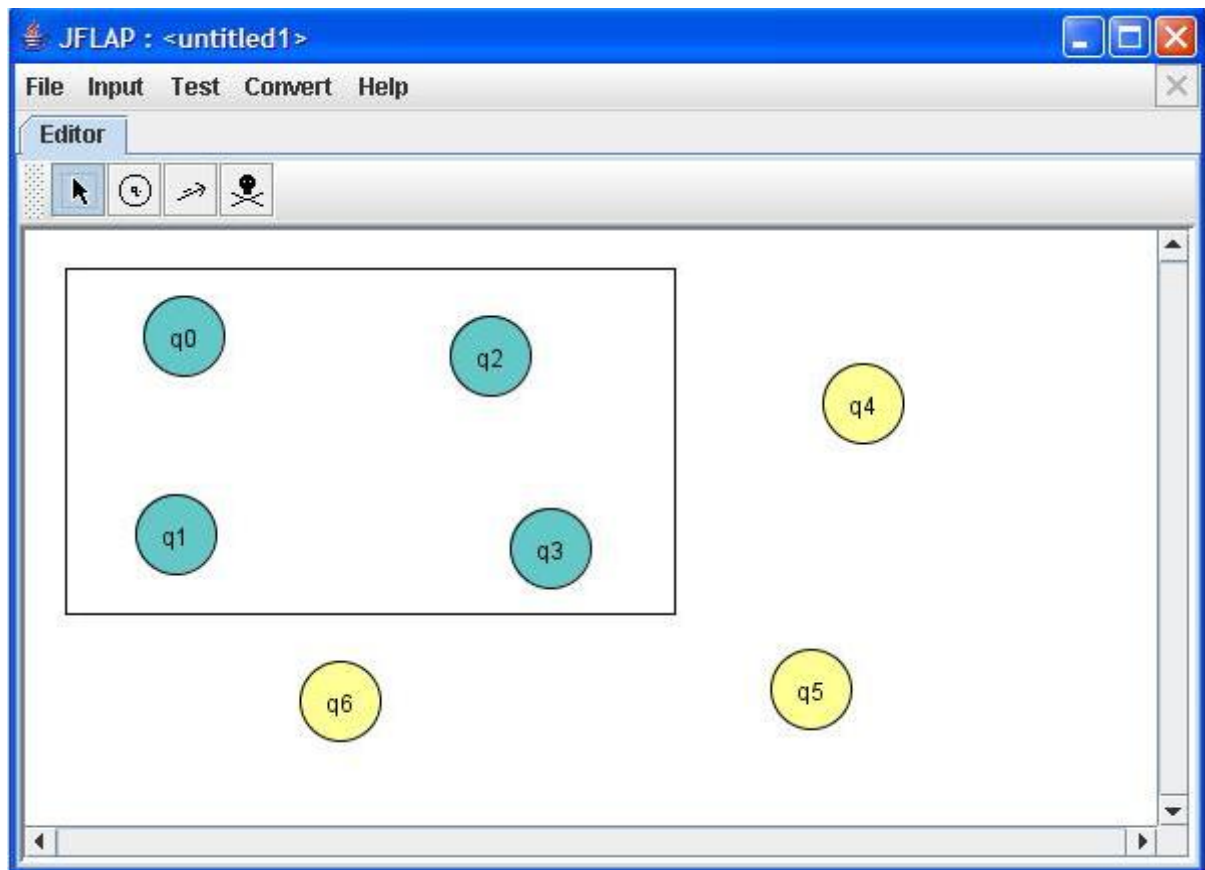
Notes

To add a note to a JFLAP file, choose the Attribute Editor, right click and select "Add Note". The note will start with the message "insert_text". To change the text simply click in the note, select where you want to start typing, and type your note. Click outside the note to get rid of the cursor. Click and drag the note to move it.



Selection

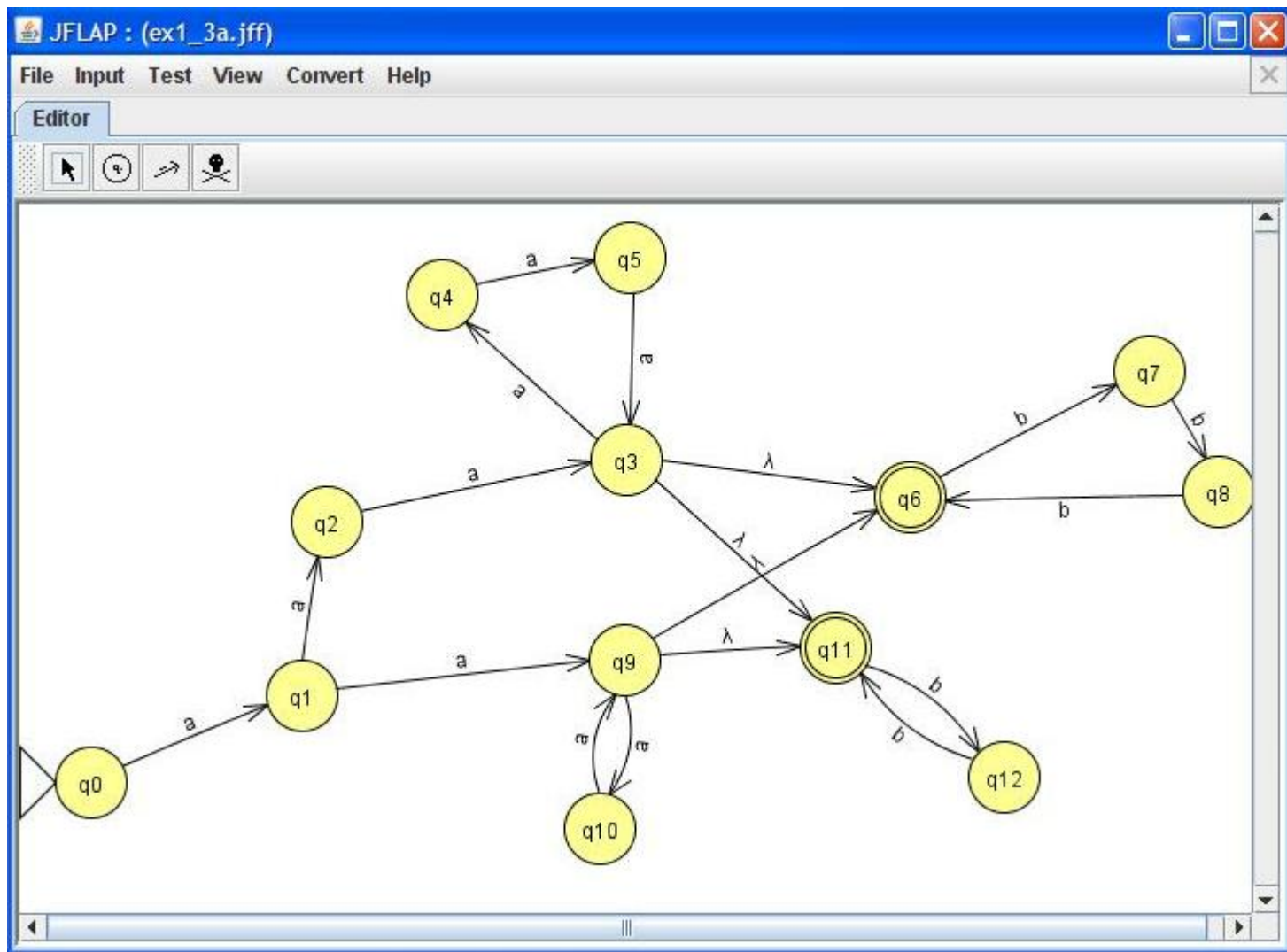
To select more than one state or block at once, choose the attribute editor, click on empty space, and drag the mouse. A bounding box appears and all states and blocks within the box are selected, their color now blue. To move the selected states as a group, click and drag any of them. To deselect them, click anywhere else.

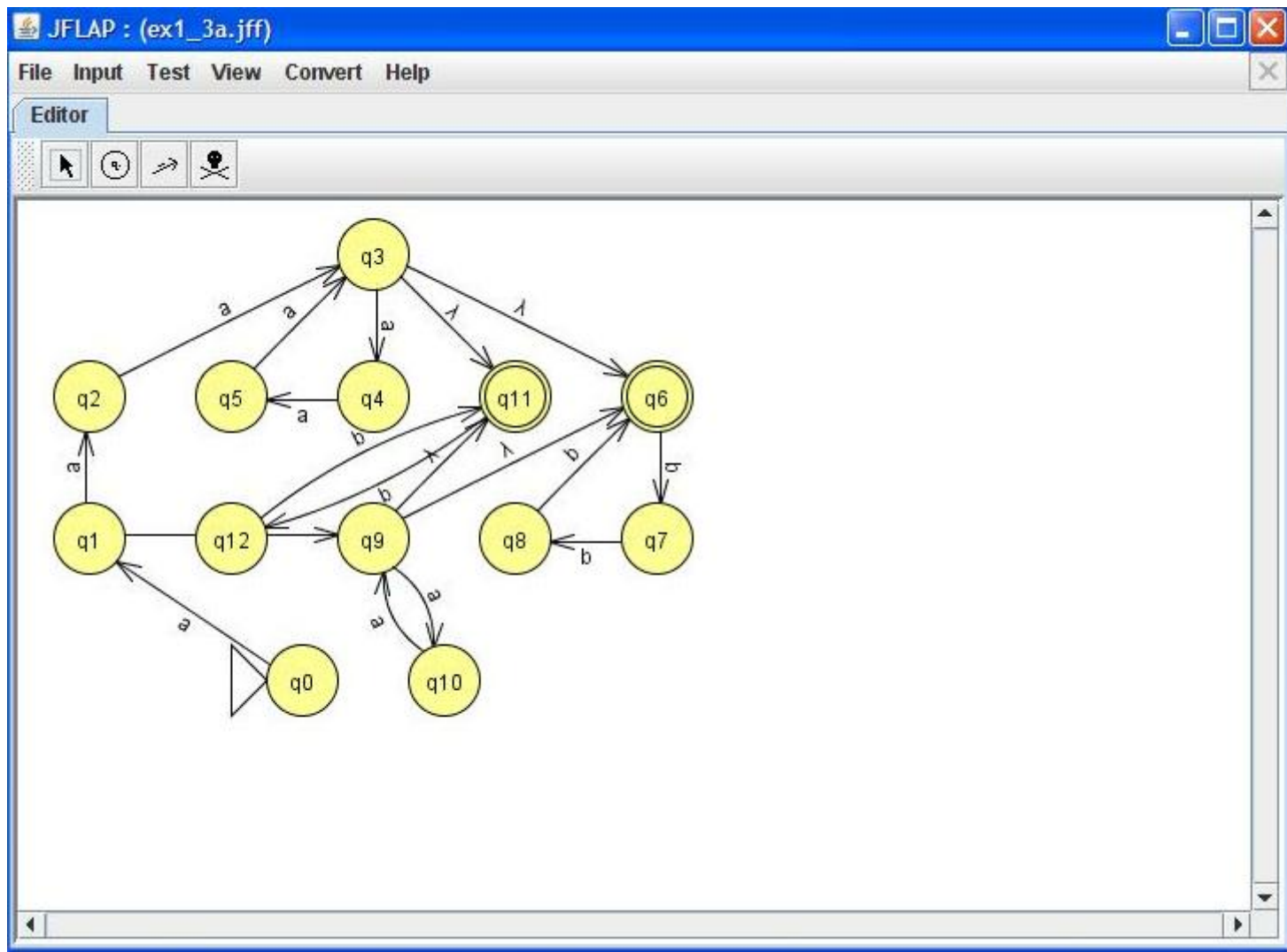


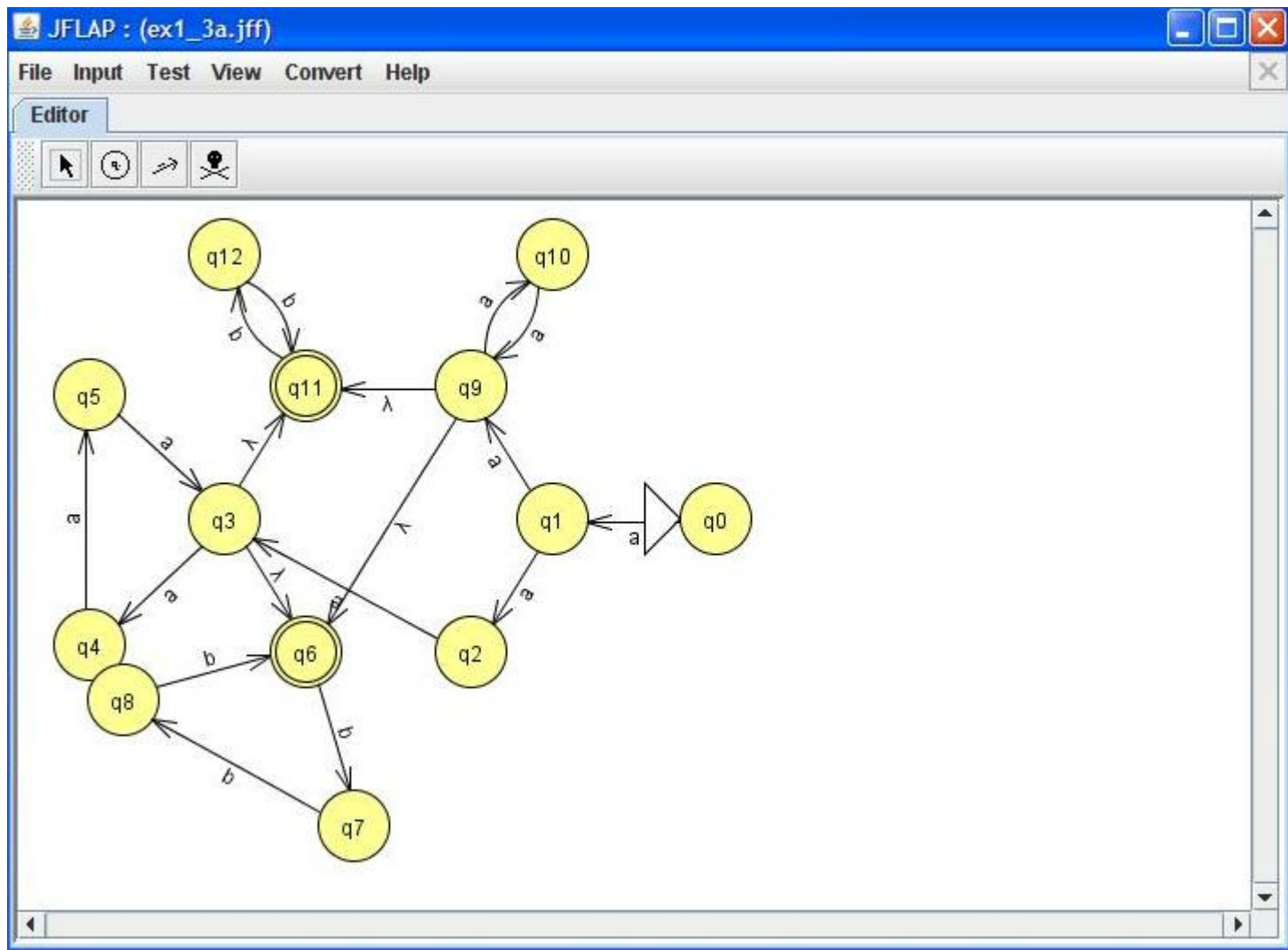
Layout Commands (as of JFLAP version 6.2)

JFLAP will now let you apply predefined graph layout commands to your graph, which can help with a more aesthetically pleasing graph. There is a new menu in the automaton editor window, the “View” menu, which will allow one to both save the current graph layout and to apply different graph layout commands and algorithms. For a full tutorial on how to use these features, and to see a description of the built-in layout commands, feel free to read the [layout command tutorial](#).

The following are pictures of the finite automaton used earlier, [ex1.3a.jff](#), with new graph layouts. The first picture demonstrates the automaton under a “GEM” layout algorithm, the second under a “Tree (Degree, Vertical)” layout algorithm, and the last under a “Two Circle” layout algorithm.







This concludes our brief tutorial on building finite automata.

Manipulating Transitions (JFLAP 7.0)

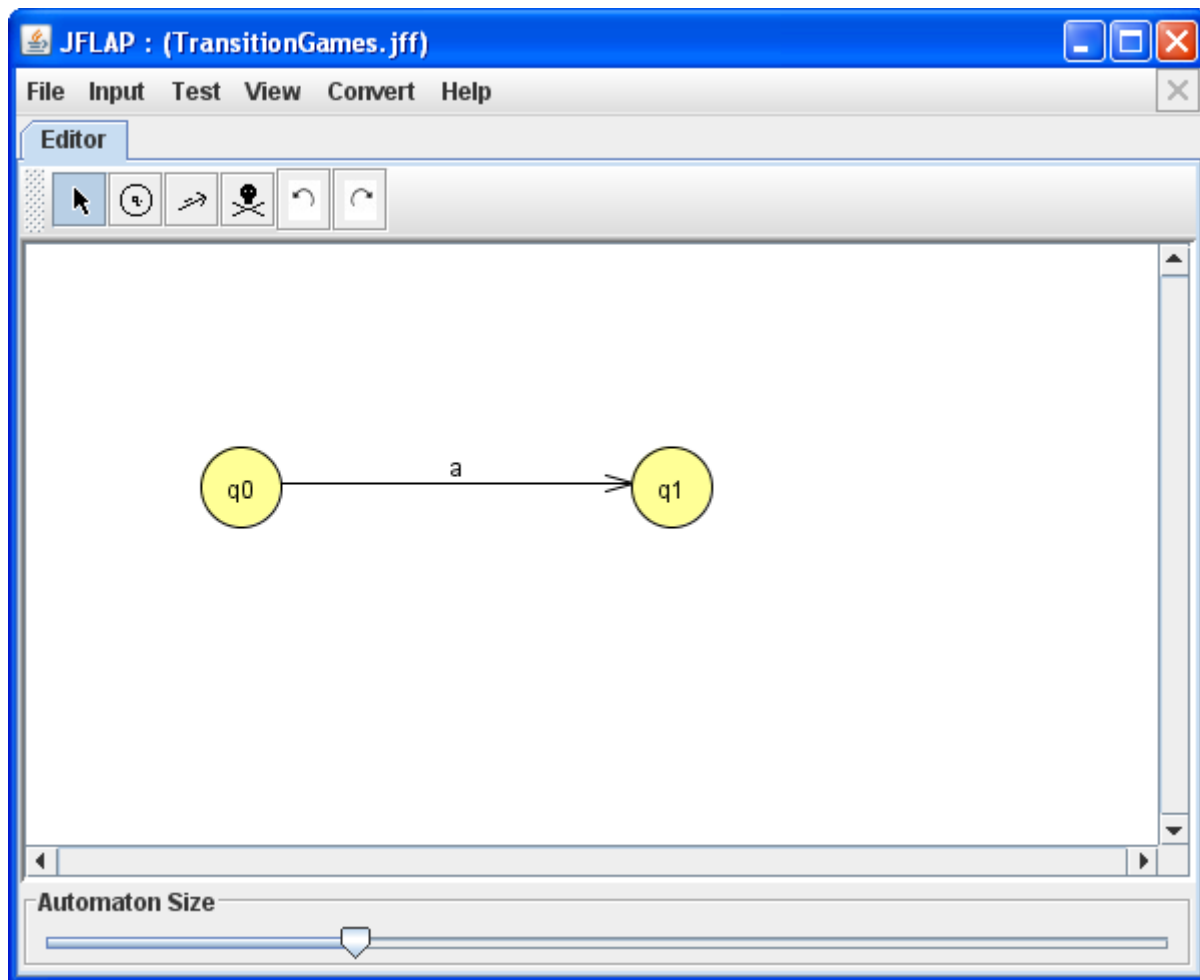
In JFLAP 7.0, we introduce the ability to customize the curvature of every transition.

If you read this tutorial, you can take advantage of this ability in a frustration-minimal way. In this tutorial, we treat a finite automata specifically, but it works the same in all the machines.

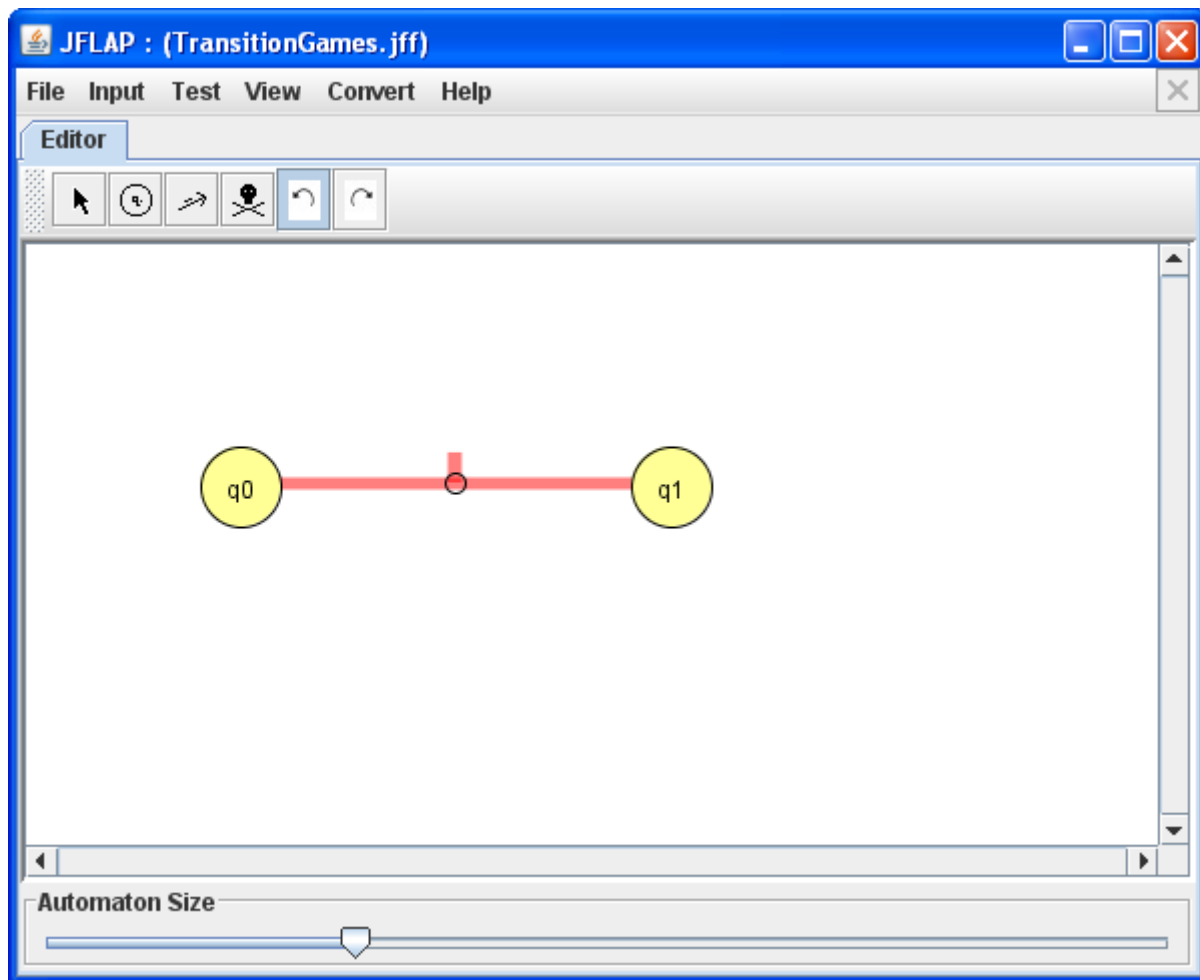
Also, note that all manipulations are done with the Arrow Tool.

Simple Case

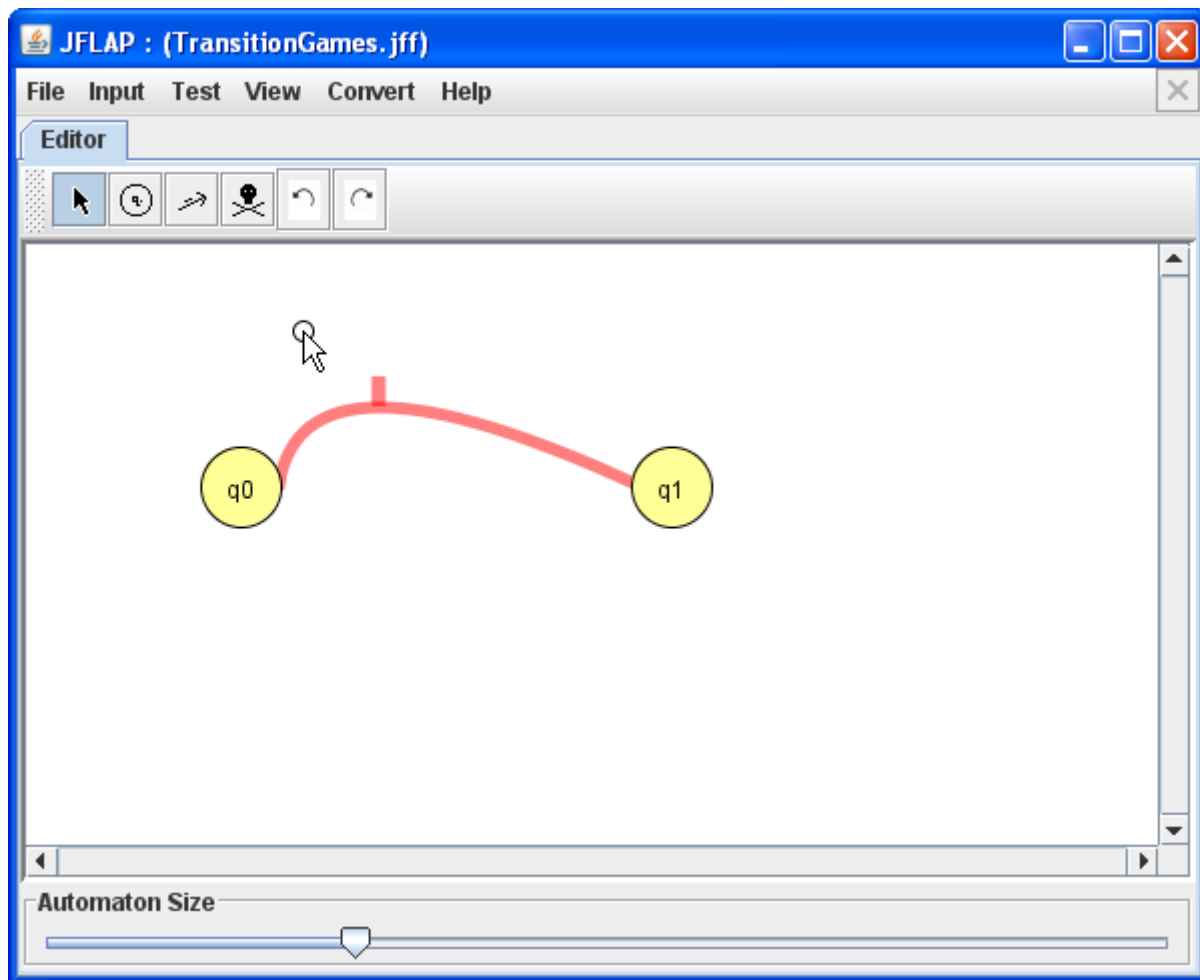
Let's start with a two-state automata, with a simple transition:



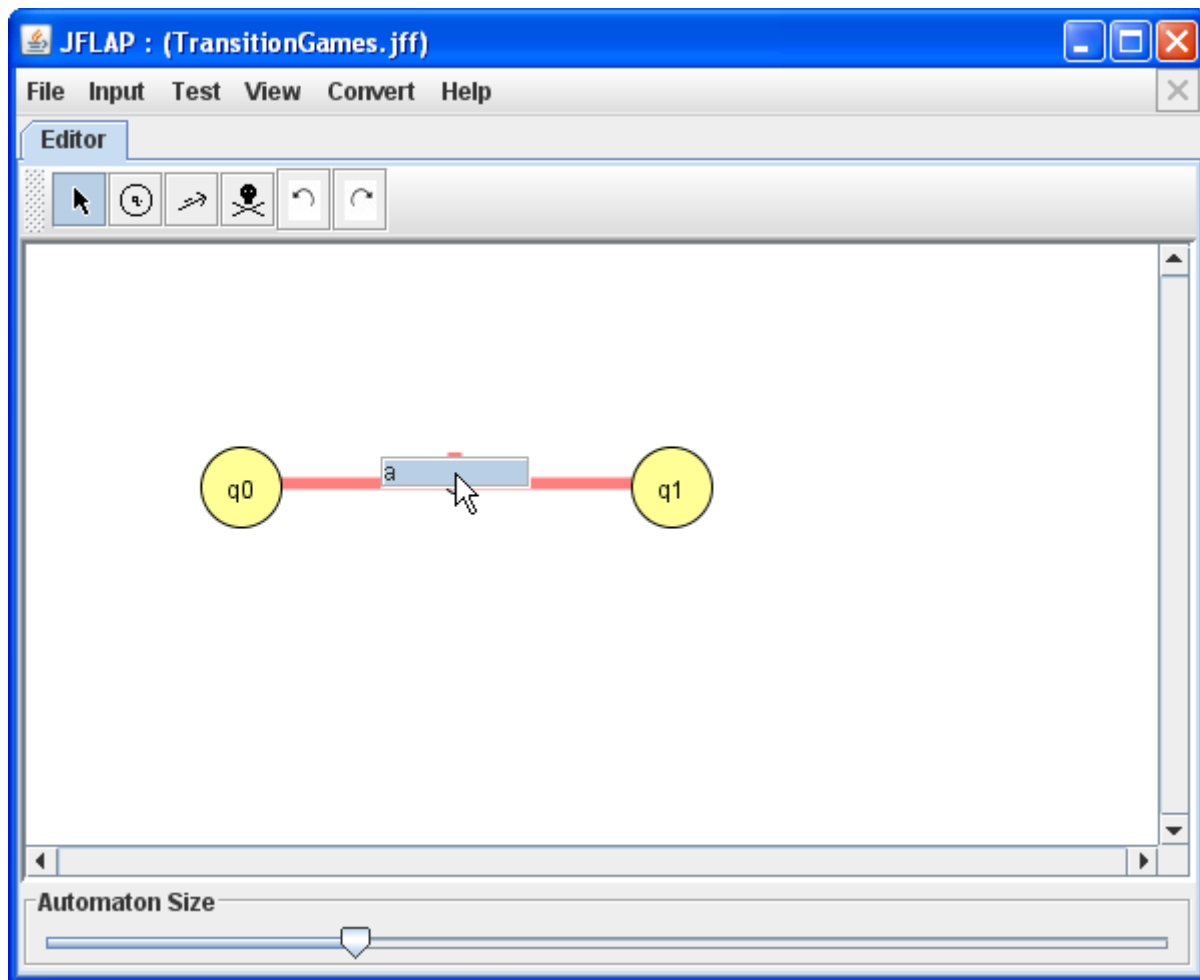
In order to manipulate the transition, click once on the transition so that it is selected. This is indicated by the highlighted color.



Once the transition is selected, the transition can be manipulated by clicking inside the circle and dragging.



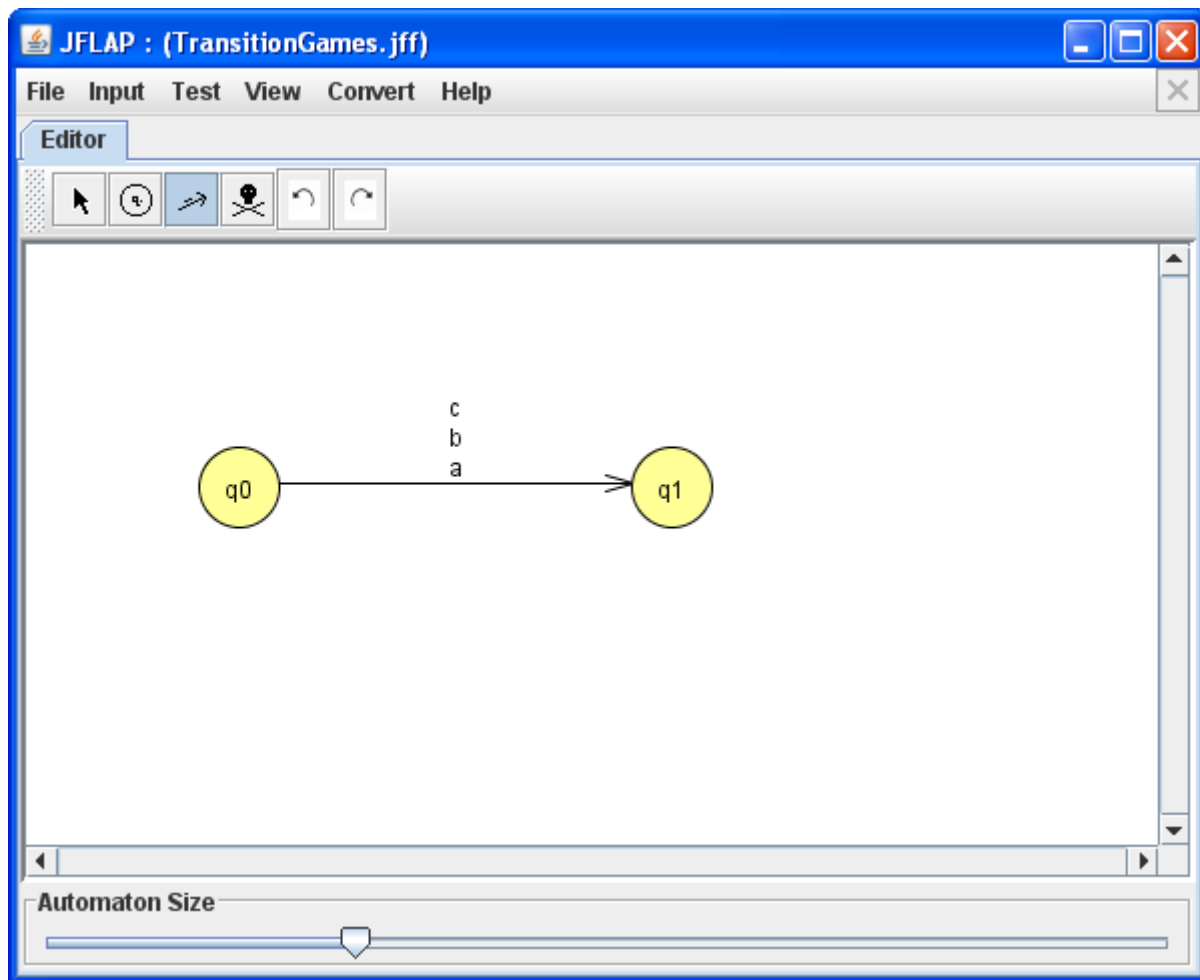
In order to EDIT the transition, double-click on it. Note that in JFLAP 6.4, it was only a single click to edit.



You will observe that the double-click will also have the effect of selecting the transition. You can de-select it after you're done editing, by single-clicking once more.

Complex Case - Multiple Labels on a Transition

If there are multiple labels on a transition, as below:



Then each label is independent, with respect to transition manipulation. We can select on each label (by clicking on it), and the control point for THAT label will be made visible. The label that is closest to the visible transition is bound to it.

