Building A Turing Machine

Contents

Definition

JFLAP defines a Turing Machine M as the septuple $M = (Q, \Sigma, \Gamma, \delta, q_s, \square, F)$ where

Q is the set of internal states $\{q_i \mid i$ is a nonnegative integer$\}$
Σ is the input alphabet
Γ is the finite set of symbols in the tape alphabet
δ is the transition function
S is $Q * \Gamma^n \rightarrow$ subset of $Q * \Gamma^n * \{L, S, R\}^n$
$\square$ is the blank symbol.
q_s (is member of Q) is the initial state
F (is a subset of Q) is the set of final states

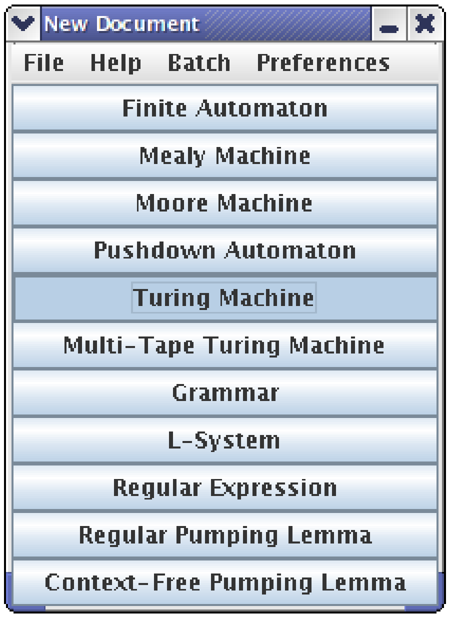
Note that this definition includes both deterministic and nondeterministic Turing machines.

How to Create a Turing Machine

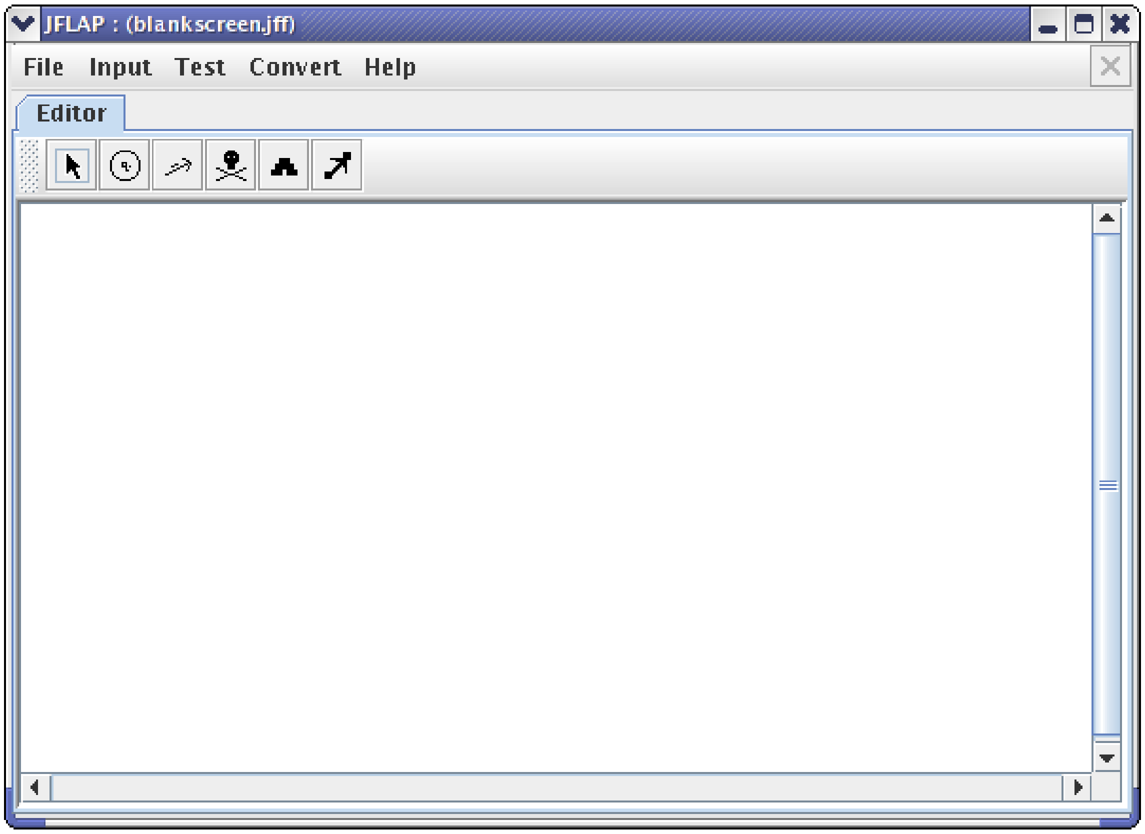
For knowledge of many of the general tools, menus, and windows used to create an automaton, one should first read the tutorial on finite automata. This tutorial will principally focus on features and options differing from the finite automaton walkthrough, and offer an example of constructing a Turing machine.

Before starting, click on the “Preferences” item in the menu. A few preferences will be listed, and one of them, with a check box next to it, is “Enable Transitions from Turing Machine Final States.” Don't do anything with this preference just now and leave it unchecked, but just note that it exists.

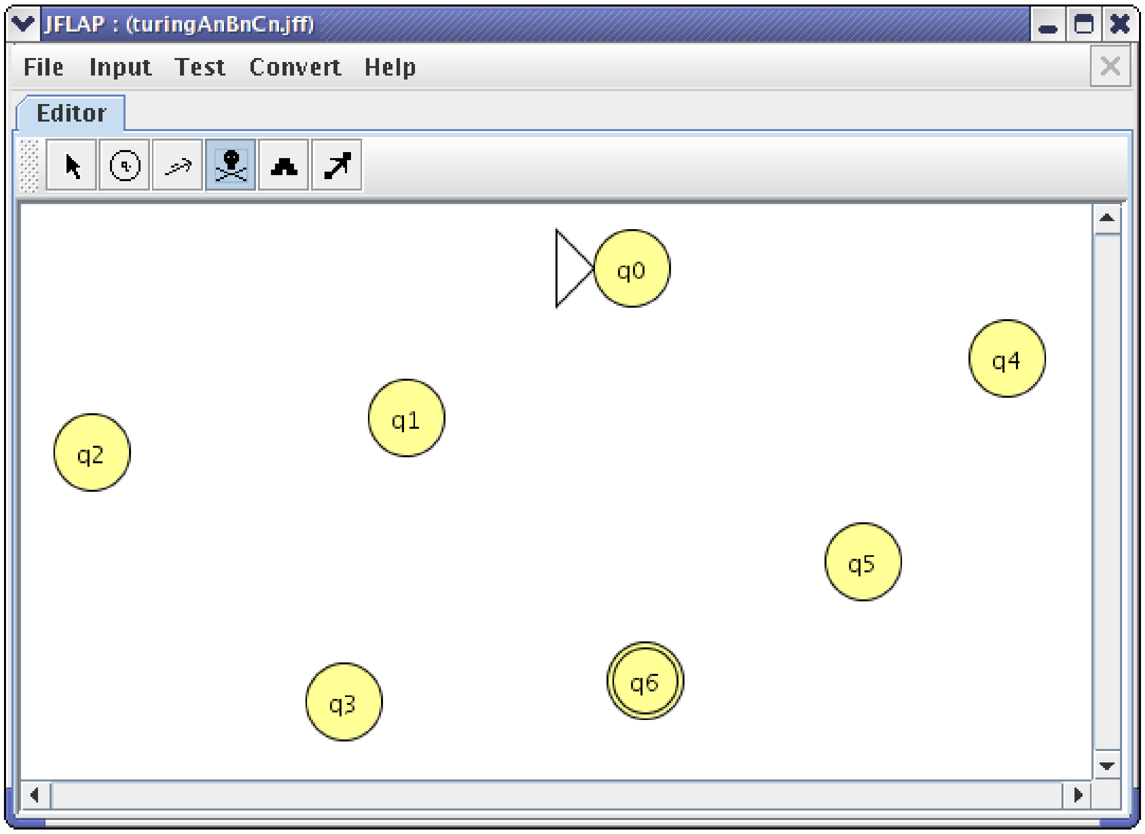
We will begin by constructing a Turing machine for the language $L = \{a^n b^n c^n\}$. To start a new one-tape Turing machine, start JFLAP and click the **Turing Machine** option from the menu, as shown below:



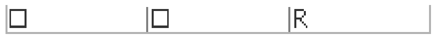
One should eventually see a blank screen that looks like the screen below. There are many of the same buttons, menus, and features present that exist for finite automata. This tutorial will principally focus on features and options that differentiate Turing machines from finite automata.



We will be adding a lot of states to create a Turing machine for $L = \{a^n b^n c^n\}$. Add seven states to the screen, setting the initial state to be q0 and the final state to be q6. The screen should be roughly similar to one below.

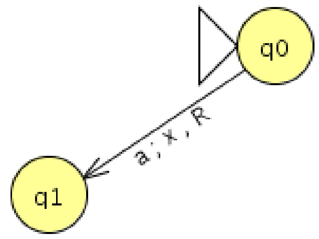


Now its time to add the transitions. Attempt to add a transition between the states q0 and q1. However, there is something different about these transitions in comparison to those created with finite automata. One will notice that there are three inputs instead of one.

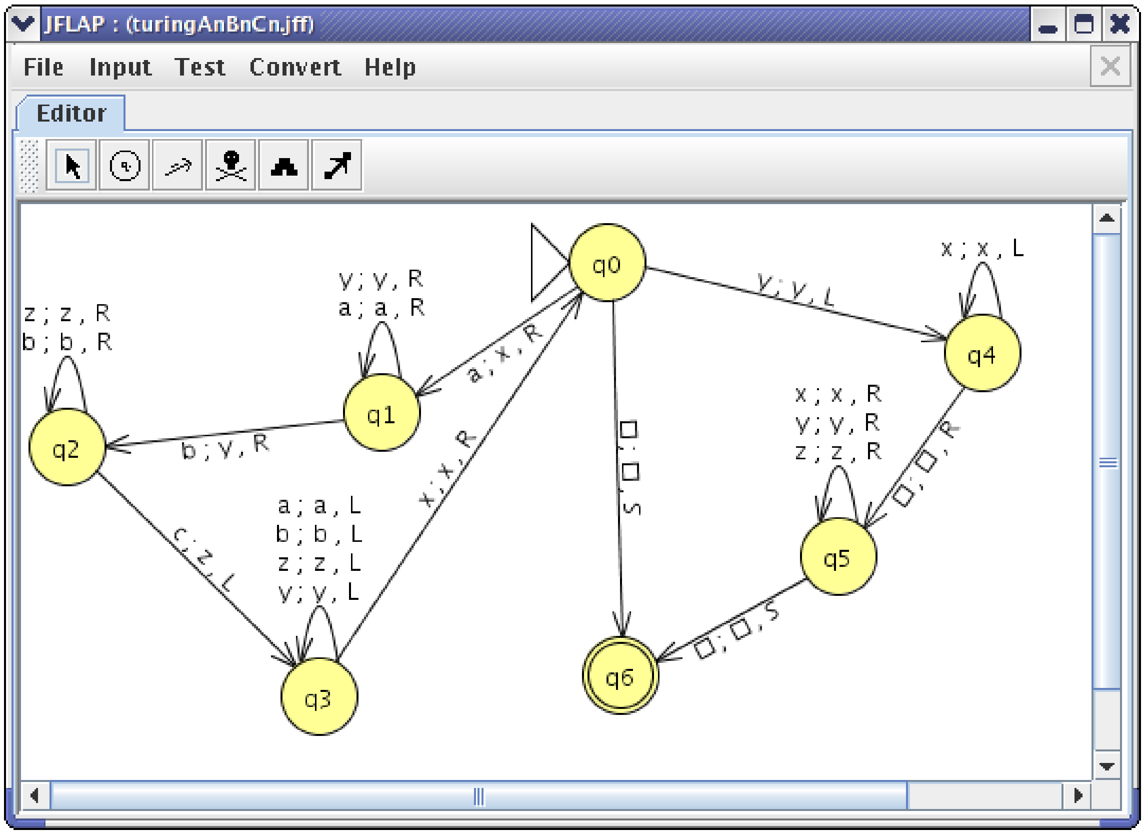


The value in the first box represents the current value under the head of the Turing machine. The second value is the value that will replace the first value on the tape once this step has been processed. The size of the values in these two boxes is limited to one character. The third value represents where the head will move after processing the step. It can be one of three values: 'R' (move right one square), 'L' (move left one square), and S (stay put and do not move the head). One could enter the value directly, or enter it from the pull-down menu that comes up when the third box is clicked on directly.

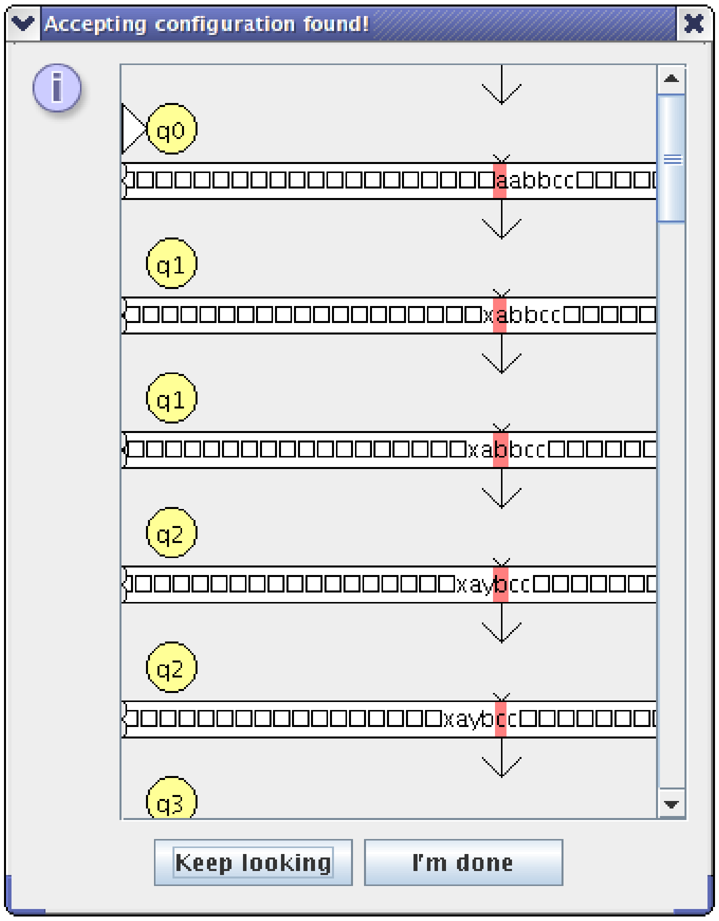
Now, it's time to add input. To change the transition from the default, click on the first box. Enter a value of “a” for the first box, a value of “x” for the second box, and a value of “R” in the third box. Use Tab or the mouse to move between the boxes, and press enter or click the mouse on the screen outside the boxes when done. This transition has the following meaning. If the head is under an “a” and the machine is in state “q0”, then replace the “a” with an “x” and move the head to the right. When done adding input, the area between q0 and q1 should resemble the example below.



Let's finish up the transitions. Add the transitions in your screen below to your Turing machine. If you would rather not add every transition directly and would prefer to load the file of the screen below, it is available at turingAnBnCn.jff.



Now, let's try out our new Turing machine. Because of the number of steps, we will avoid the "Step" option we used with finite automata (although for finite automata titled "Step with Closure") and instead use the "Fast Run" option. To use this, click on the "Input" menu, and then click on "Fast Run". When it prompts you for input, enter "aabbcc", representing $a^2b^2c^2$. After clicking "OK" or pressing enter, the following screen should come up:

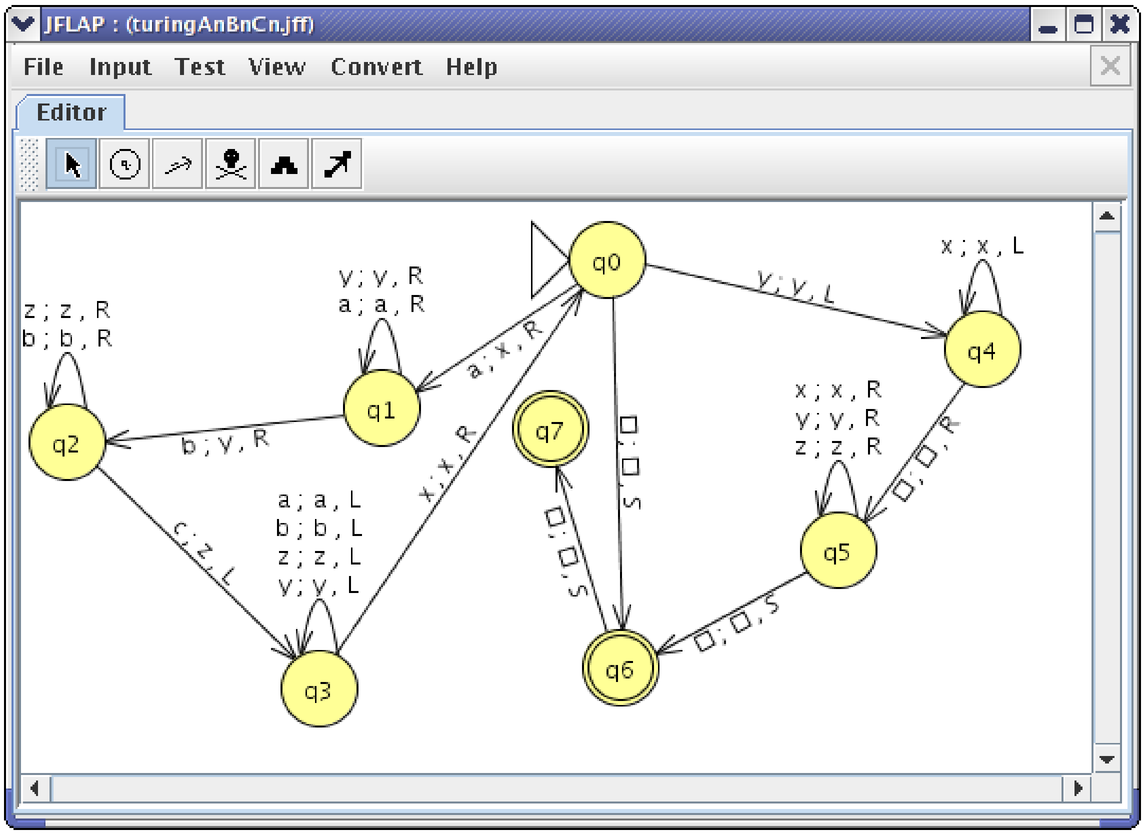


One can scroll down and see the tape, the current state, and the position of the head as the automaton processes the input step by step. One can see the algorithm at work, which is if the head encounters an “a”, it replaces it with an “x”. Then, it replaces a corresponding “b” with a “y” and a corresponding “c” with a “z”. This repeats until it is no longer possible, and this loop is what makes up the cycle encompassing “q0”, “q1”, “q2”, and “q3”. Once this is done, the program makes sure that there is nothing but “x”s, “y”s, and “z”s left in the correct order. In the case of input with length zero, the program immediately goes to the final state.

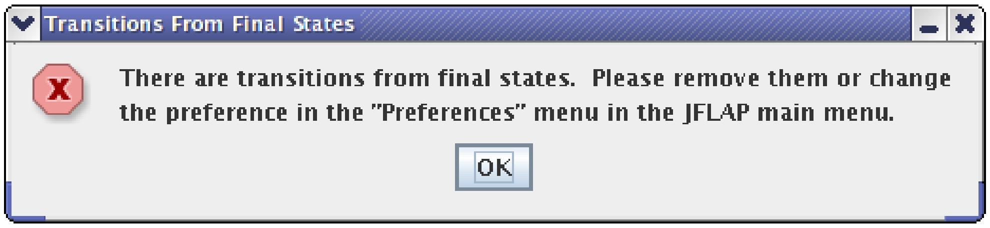
The “Keep looking” button is for finding other possible paths in automata that aren't deterministic, which is not applicable here. When finished, click “I'm done.” Congratulations, you have built your first Turing Machine!

Transitions from Final States

Recall the “Enable Transitions from Turing Machine Final States” preference mentioned earlier. Also note the slightly modified earlier example below that now has two final states.



Because the preference was not enabled, and because there is an edge leading from the final state “q6”, the following error message will appear if you try to run it. Just note that if you wish to simulate such a machine, you need to either enable the preference or remove all offending edges.

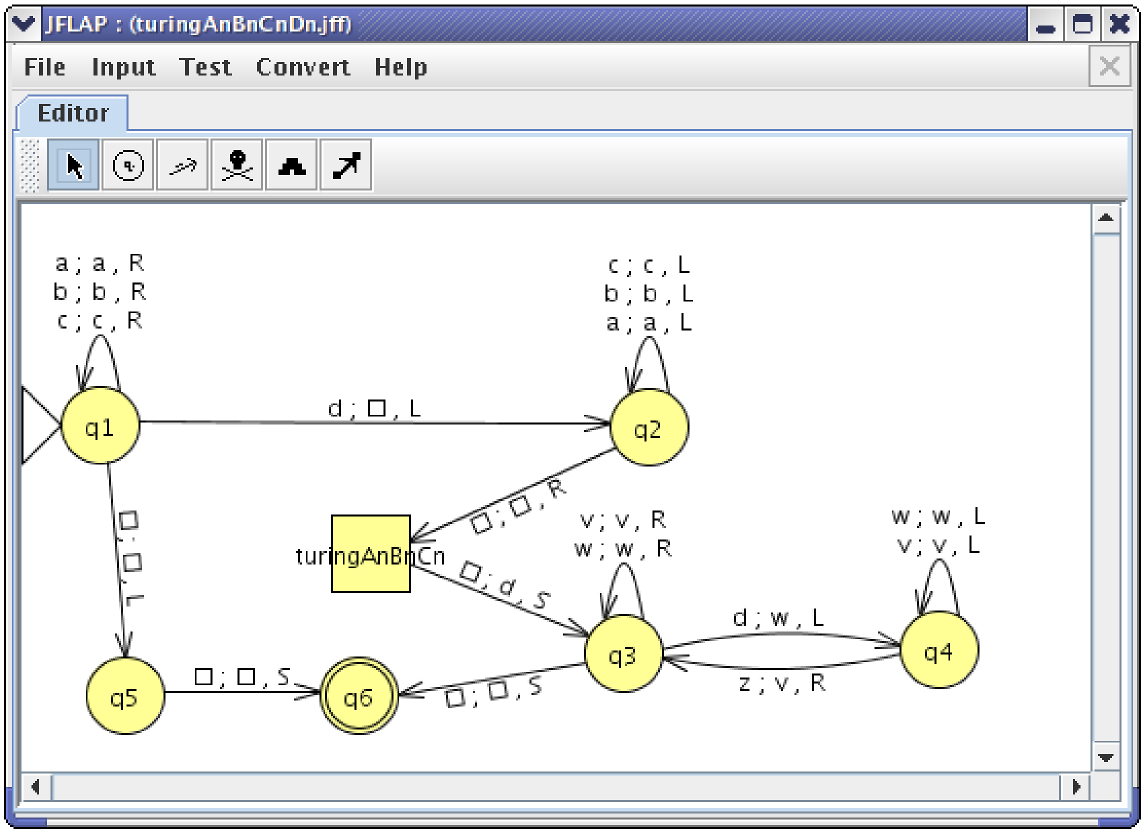


There are transitions from final states. Please remove them or change the preference in the "Preferences" menu in the JFLAP main menu.

Using Your New Turing Machine as a Building Block

There are a few other features in JFLAP concerning Turing machines, and one very useful one is “building blocks”. We will not go into an in depth study of building blocks on this page, and one can learn more about them here. However, it is worth noting that Turing machines, once created, can function as building blocks in other machines. Below is one such example, where the block we just created for $L = \{a^n b^n c^n\}$ is used to implement the language $L = \{a^n b^n c^n d^n\}$. One can create Turing machines to accomplish one task, and if another task could utilize the first task to further its designs, one can use a building block as a shortcut to represent that task on the screen. The “block” was put onto the screen by the second rightmost button in the toolbar, which has an icon resembling a step pyramid. When clicked, a file menu will come up, as if you were opening a file. By selecting the file for the language $L = \{a^n b^n c^n\}$, and clicking “Open”, a yellow square will appear on the screen labeled by the file name. This functions on the screen as a state, and transitions can be created to and from it. Whenever something leads to the building block, it will preform its task based on the current state of the tape, and the tape will be changed by the block's output, for the benefit of the rest of the states in the machine.

The screen below is an example of using a block for $L = \{a^n b^n c^n\}$ as a tool to implement the language $L = \{a^n b^n c^n d^n\}$, this example is accessible here:



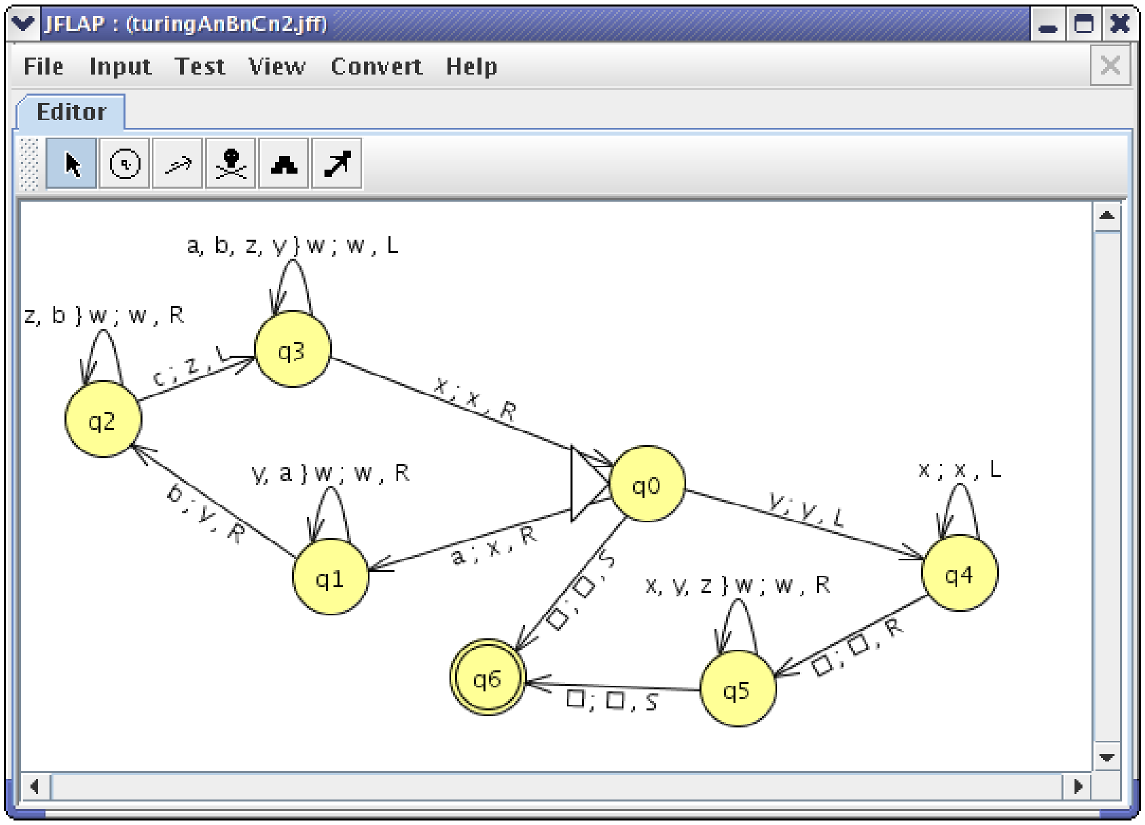
One may try a variety of different inputs and realize that the block functions as an acceptor by checking to see whether the number of “a”s, “b”s, and “c”s equal at the beginning of the input. It also functions as a transducer, as it changes those values to “x”s, “y”s, and “z”s respectively. The program builds off of this by checking to see whether the number of “d”s equals the number of “z”s. One must also note that a blank was placed where the first “d” value was. This is because, in order for the $a^n b^n c^n$ acceptor to work, one must have blanks surrounding the smaller string. The “d” value is restored after leaving the block.

Shortcut Syntax for Turing Machines

It is worth mentioning a few shortcut syntax rules that JFLAP implements. These syntax rules were developed for use with building blocks, but they also can be used with standard Turing Machines. These rules are covered in depth in the building blocks tutorial, but for the sake of a general overview, a brief summary is as follows. The first shortcut is that there exists the option of using the “!” character to convey the meaning of “any character but this character.” For example, concerning the transition (!a; x, R), if the head encounters any character but an “a”, it will replace the character with an “x” and move right. To write the expression “!□”, just type a “!” in when inputting a command.

One can also utilize variables when constructing a Turing machine in order to make inputting rules less tedious. For example, there is a second special character, “~”, that stands for whichever character was last read. Thus, in the transition (~; ~, R), the head will, no matter what character is underneath it, move to the right without changing the character. One can also explicitly define other variables. For example, the transition (a,b,c}w; w, R) would command the head, if either “a”, “b”, or “c” was under it, to assign the letter to the new variable *w* and then to move right without changing the tape. Whenever w is encountered later in the machine, it is synonymous with its stored value until it is assigned another value.

Our Turing machine from earlier is shown below in a slightly different layout with some variable-containing transitions (there was not a good place to use the “!” feature). Notice the fewer transitions present for preforming the exact same task. This machine is available in turingAnBnCn2.jff.



This concludes our brief tutorial on building Turing machines. Thanks for reading! If you wish to learn about implementing Turing machines with more than one tape, click here.