Creating Finite State Machines using JFLAP

Courtesy of Troy Vasiga `troy.vasiga@uwaterloo.ca`
Carmen Bruni `cbruni@uwaterloo.ca`
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, N2L 3G1
CANADA

1 Acknowledgments

This worksheet would not be possible without JFLAP, and JFLAP would not be possible without Susan Rodger at Duke University.

The latest version of JFLAP can be found at:

`http://www.jflap.org`

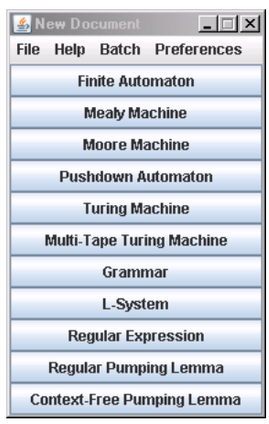
2 Introduction

In lecture, we discussed *Finite State Machines* (also known as *FSMs, finite state automata,* or *FSAs*).

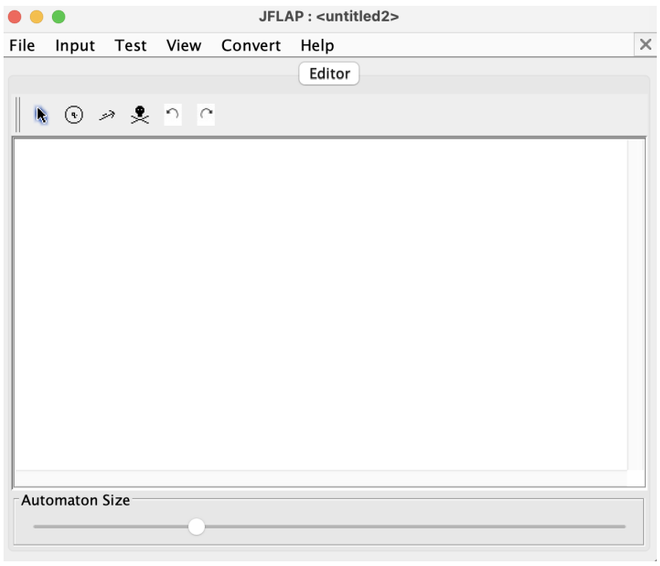
3 Getting Started

We outline how to launch JFLAP.

1. Log-on to your account.
2. Click on the three-by-three grid of squares in the bottom left of your screen, click on the box near the magnifying glass in the top middle of the screen and type `JFLAP`.
3. Click on the item that you found. After a little while, a window looking a lot like the following window will pop up:



4. Click on “Finite Automaton.”
5. You should see the following screen



You are now ready to create some FSMs.

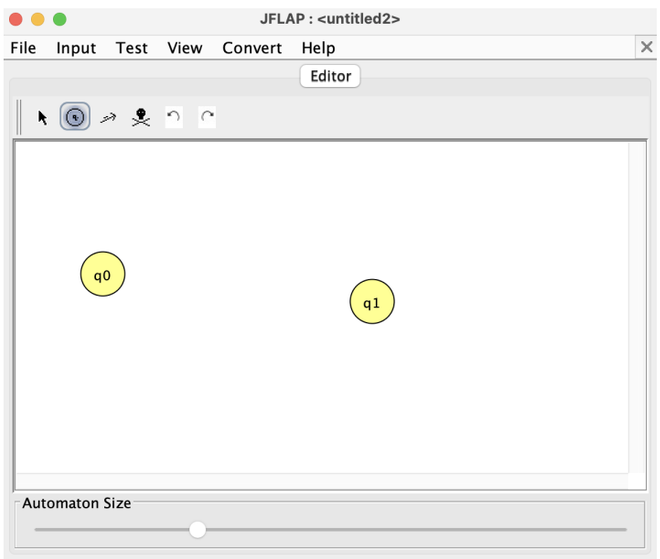
4 My first FSM

Let's begin by creating a machine which accepts the string `a` and only that string.

4.1 Creating States

1. In the JFLAP window, click on the "state" (circle) icon, which is the second element.

2. Move the mouse to the main window, and click twice in two different positions. You should get two states, one labeled `q0`, the other labeled `q1`. The picture should look something like the following:



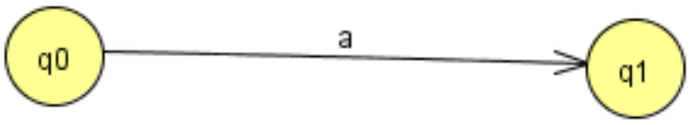
4.2 Adding transitions

We now add an arc (transition) between the two states.

1. Click on the third button in the edit window (it is shown as selected here).

2. Click on state `q0` with the left mouse button, and *while holding down the button*, drag over to state `q1`. Release the left mouse button.
3. A small text box will appear, as shown in the following diagram.

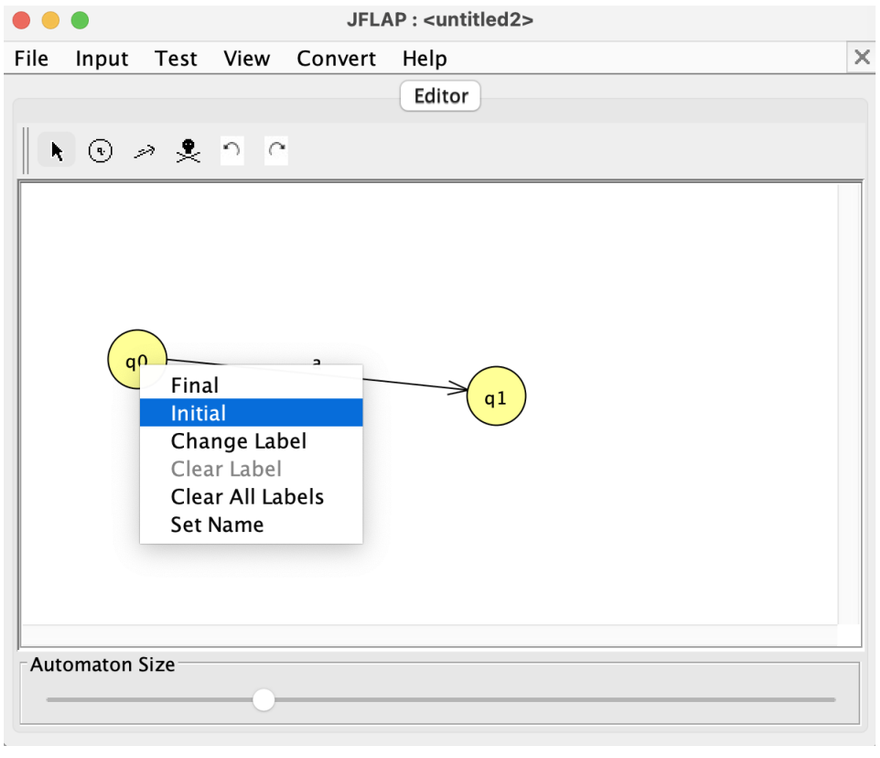
4. Click on the box with the left mouse button, type the letter `a` and hit Enter twice.
5. The diagram should look like this:



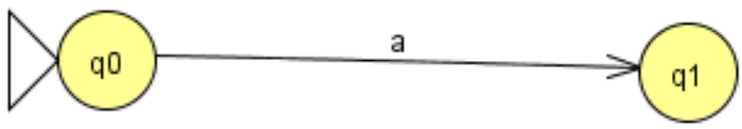
4.3 Indicating start states and final states

1. First, click on the Attribute Editor (left most button in the Editor panel). It is shown below.

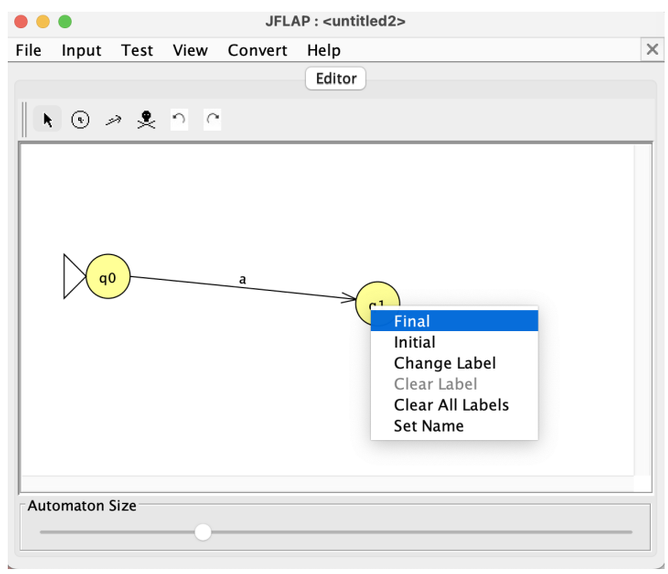
2. Next, right click (or hold down the `control` key while clicking, if you are using a one-button mouse) on state `q0`. A small window will appear. Select (using the left button) the attribute that you wish to set. In this case, set this state as the Initial (start) state.



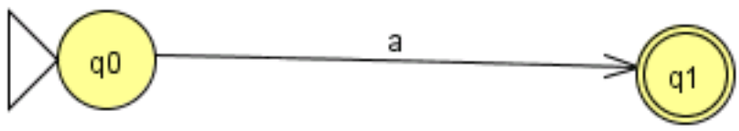
3. Once you have done this, the machine should look like the following:



4. Next, we add a final state in the same manner. With the Attribute Editor selected, right click (or `control` click) on state `q1` and select "Final", as shown below.



5. Once you have selected "Final", the picture should look like the following:



5 Saving your work

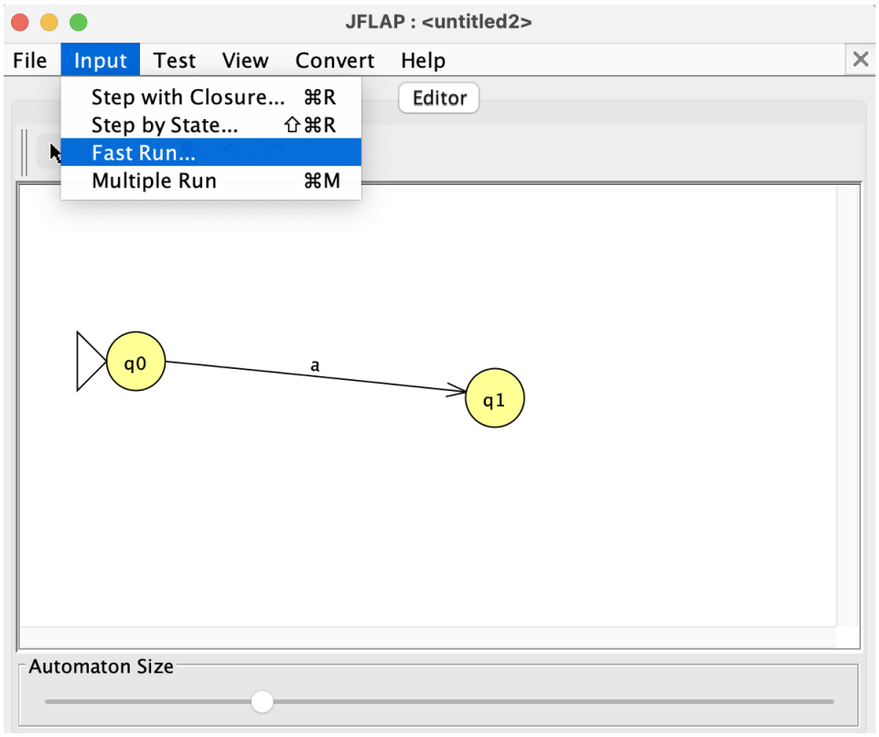
Now would be a good time to save your work. Simply click on "File" and select "Save" to save your file. In the default directory, create a "New Folder" (called `JFLAP_files`).

Call this file `fsm01` (which JFLAP will save as `fsm01.jff`) and then click "Save".

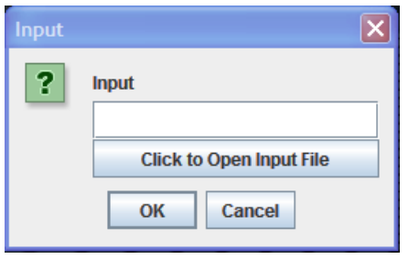
6 Reading input through your FSM

We are now ready to see what words our FSM accepts or rejects.

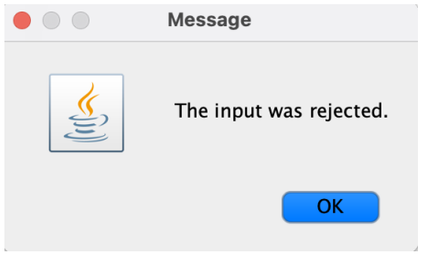
1. We will first try the input `aab` on the FSM. First, click on the "Input" menu item and select "Fast Run".



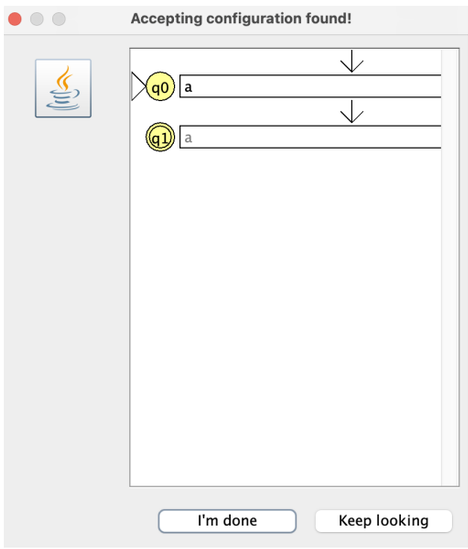
2. A dialog box will appear.



3. Type in `aab` and hit enter. This input will be rejected, and a pop-up window will indicate this:



4. To see what happens when an acceptable input is entered, repeat the above steps, except input the string `a` when prompted. The pop-up window that is created this time is the following:



indicating that the input was accepted.

5. Click "I'm done" when you are finished.

7 Fancy stuff

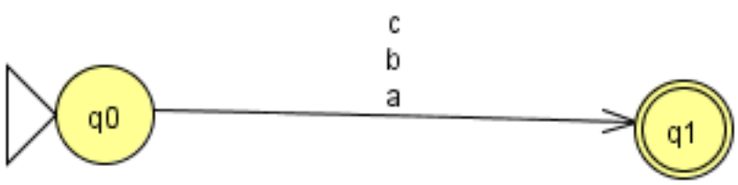
7.1 Multiple Transitions

To add multiple transitions, simply add a transition between states.

For example, if we have our `fsm01` from earlier, we can add more transitions by:

1. Clicking on the Transition Creator tool
2. Dragging from one state to the next.
3. Entering the appropriate transition in the dialog box.

Notice that the multiple symbols are stacked upon each other, as shown below, which is the `fsm01` FSM with symbols `b` and `c` added.



Try to create this FSM, and make sure that only the strings `a`, `b` and `c` are accepted. (Hint: use your "Fast Run" again.)

7.2 Deleting Things

To delete things, select the Deletion Tool (the "skull and crossbones") and click on the item to delete. The icon is shown below:

You can delete:

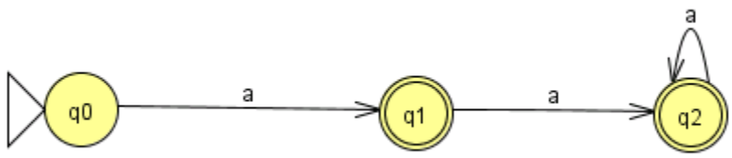
- transitions (by clicking on the arc itself)

- an element on a transition (by clicking on the symbol)
- a state and all the transitions which go into or out of the state (by clicking on the state itself)

7.3 Minimizing DFAs

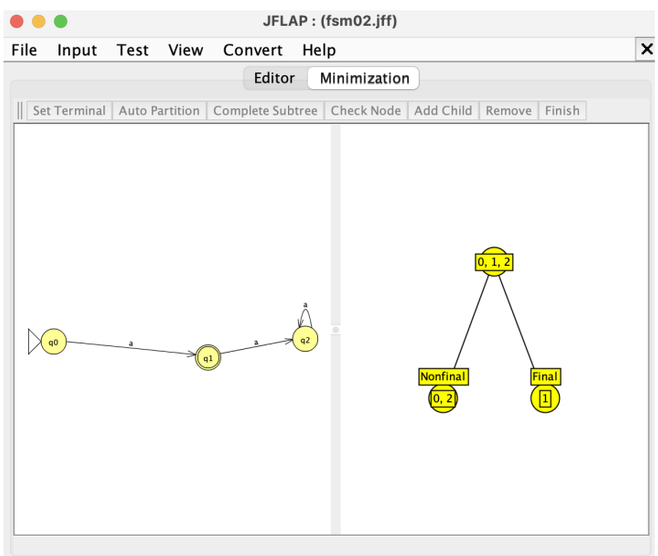
Let's suppose we have a DFA. Can we write it in a smaller format?

Let's begin with the DFA shown below.



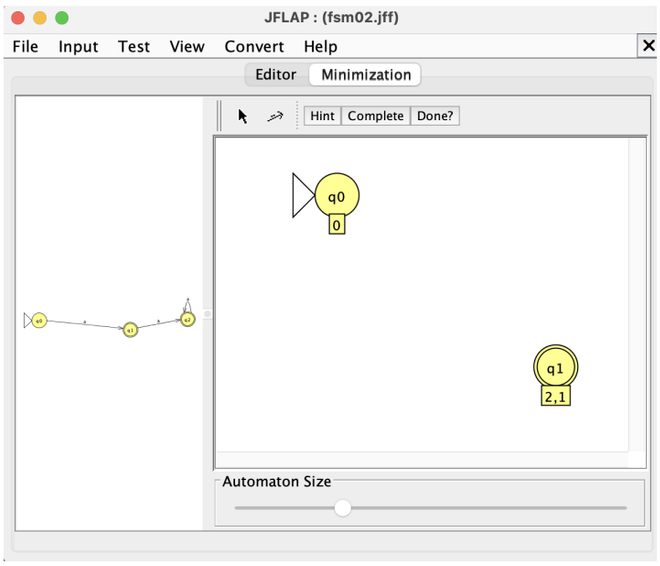
You should think about which regular expression this is equivalent to. Notice this DFA has 3 states, and two of them (`q1` and `q2`) are final. There are transitions from `q0` to `q1` and from `q1` to `q2` and from `q2` to `q2`, all on symbol `a`. (To create a "loop" transition from a state to itself, you should click twice on the same state.) Once you have entered this FSM into JFLAP, you should save it as `fsm02`.

1. Click on "Convert" and select "Minimize DFA". You should get the following picture.

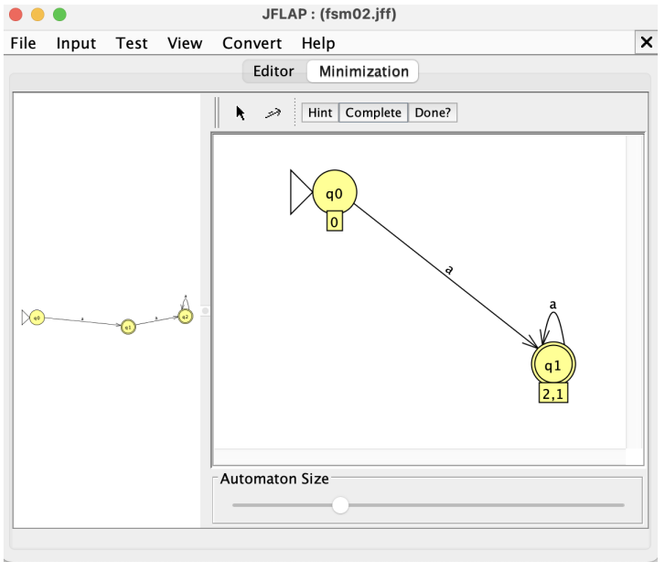


2. At this point you will notice this is a new tab. Should you wish to "Dismiss" this tab, you must select "Dismiss Tab" from the "File" menu.

   Instead, we will click on the "Finish" button to let the machine do the minimization. We get the following picture.

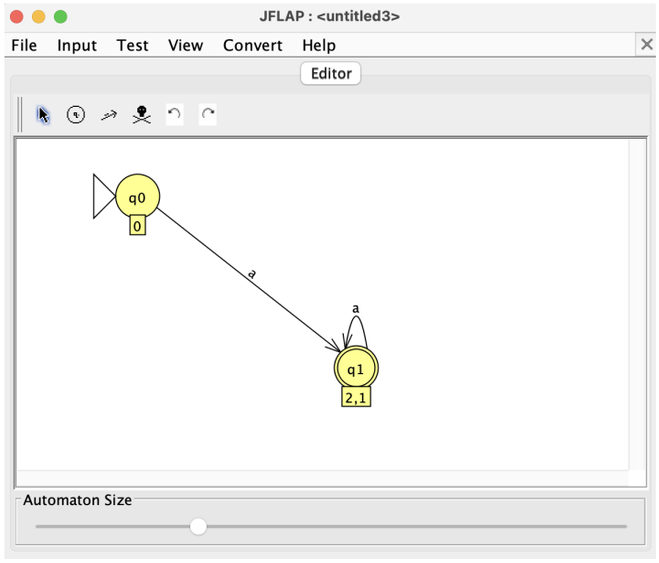


3. We want to complete this process, so we click "Complete" and all of the arcs are put in, as shown below:



Had we wanted to, we could have attempted to do the minimization ourselves, by using the Hint feature or by adding our own arcs to the diagram.

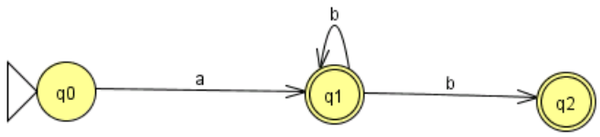
4. Since we don't want to do all that work, we click on "Done?" to get the following window (after JFLAP has informed us this window will pop up).



You should notice this DFA is equivalent to the DFA we started with (that is, it accepts exactly the same set of words as the original DFA) and it has fewer states. You should save this last FSM as `fsm03`, and then compare inputs on both `fsm02` and `fsm03` to make sure they really are equivalent. (Hint: try various sequences of `a`'s.)

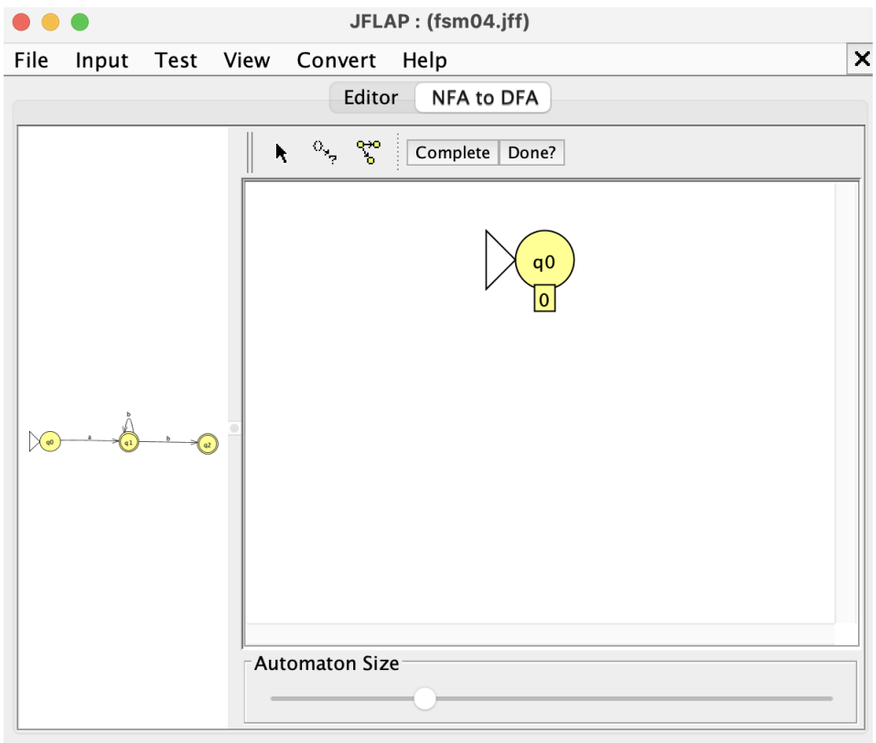
7.4 Converting NFAs to DFAs

Sometimes, we might have an NFA which would like to make into a DFA. Suppose we have the following NFA shown below (notice it is nondeterministic since if we are in state `q1` and read the symbol `b`, we could be in state `q1` or `q2`).

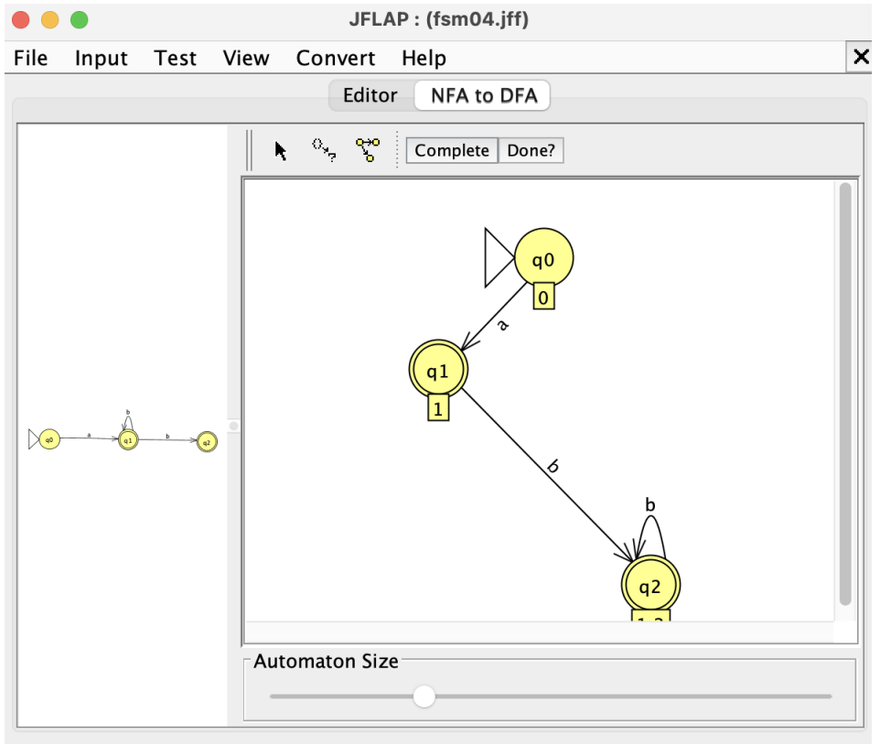


Create this NFA and save it as `fsm04`.

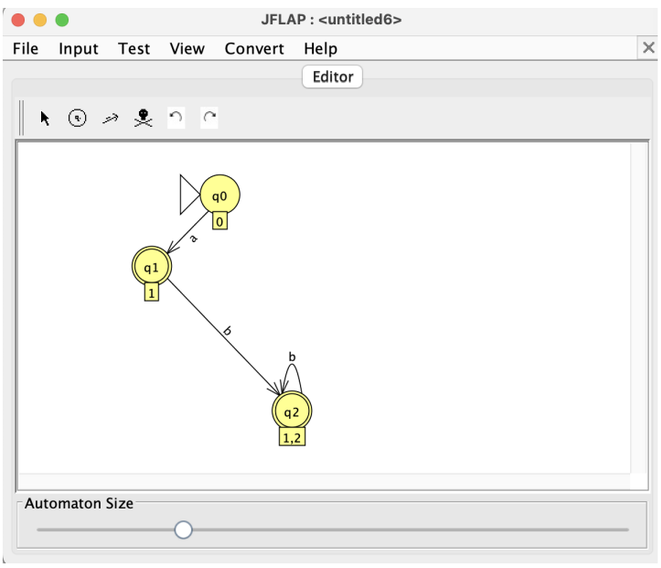
1. In a manner similar to the minimization we have completed above, select "Convert to DFA" from the menu, to get the following.



2. Select "Complete".



3. Select "Done?" and up will pop the final window, which you can verify is deterministic. Save this FSM as `fsm05` and check to make sure that both `fsm04` and `fsm05` accept the strings `ab` and `abbb`.



7.5 Converting a DFA to a Regular Expression

In order to "Convert FA to RE", you can "Do it" yourself, and when you are Done, you can "Export" the results. As an example, try to convert the `fsm03` FSM to a regular expression (which you should not be surprised to see is $\texttt{aa}^*$).

7.6 Adding λ-transitions

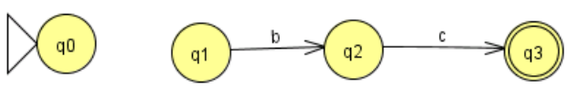
If you wish to combine two finite automata, or if you want to move from one state to another without reading input, you will need to use λ-transitions. In fact, these are quite easy to add, since this is the default character on any new transition.

Suppose we want to accept either the word `abc` or `bc`. One way to do this would be to read either `a` or nothing (λ) and then read `bc`.

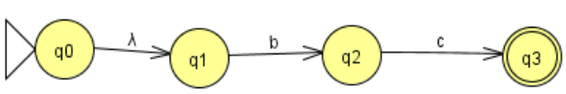
Here are the details.

1. Create 4 states, `q0` as start and `q3` as final state, as shown below.

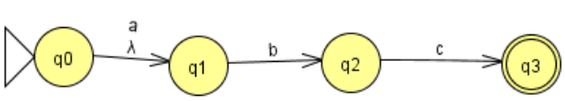
2. Create transitions from `q2 to q3` on `c`, and transition from `q1` to `q2` on `b`. You should get the following DFA.



3. Now, add a λ-transition by click and dragging a transition from `q0` to `q1`, and then hitting Enter. You will get the following picture.



4. Finally, add a transition from `q0` to `q1` on input `a` to get this final picture.



5. Try this λ-NFA on inputs `abc`, `bc` and `bcc`, to make sure it does what you think it should do.

7.7 Multiple Runs

You can also run multiple inputs on the same finite automaton using just one tool. If you:

- Click on *Input*
- Select *Multiple Run*

you can then type in a sequence of inputs that will be run on your finite automaton. Specifically, type the input you want and press Return after each input. You can type as many different inputs as you wish.

When you are ready to run all of your inputs, press *Run Inputs* and can see which inputs will be accepted and which will be rejected.

Running multiple inputs will make testing your solutions in the Exercises much easier.

8 Exercises

Try the following. See how far you get. Write down your answers (and save them to a file, if you wish). The questions are relatively ranked from easier questions at the beginning to trickier questions at the end.

1. Construct a FSM for the following regular expression:

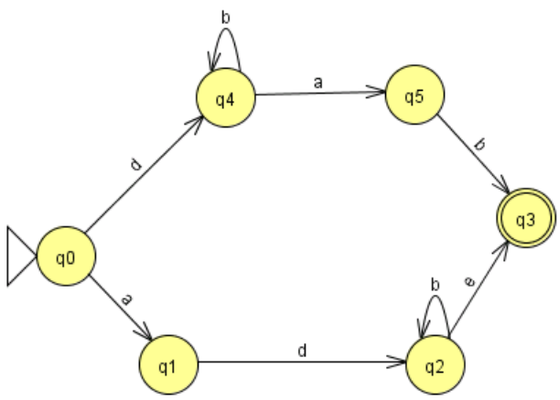
$$(ab)^*$$

2. Construct a FSM for the following regular expression:

$$a^*b^*$$

3. Construct a FSM over the alphabet $\Sigma = \{\texttt{0}, \texttt{1}\}$ which accepts only the words which have exactly three `1`'s. That is, `01010001` should be accepted, but `1111` and `010100` should be rejected.

4. What is the regular expression given by this machine?



5. Give a FSM for your postal code. For example, the postal code for the University of Waterloo is `N2L 3G1`. However, people write this a whole bunch of different ways. That is:

   - n2l3g1
   - N2L-3G1
   - N2L 3g1
   - N2L 3G1

   are all valid. Write a postal code recognizer which accepts these variations on your postal code (**HINT**: you may want to use λ-transitions.)

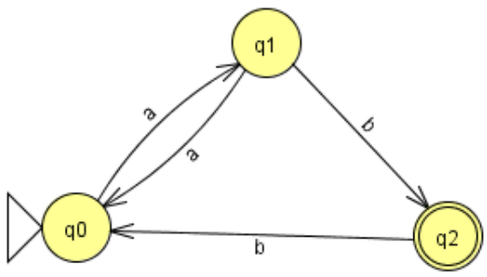
6. Construct a FSM which accepts even base-10 positive integers. For example `1020` and `2` should be accepted, but `1111` should be rejected. (**HINT:** Non-determinism will be your friend. You can do this with 2 states and 2 transition arcs nondeterministically, or with 2 states and 4 transition arcs deterministically.)

7. Repeat the previous question, except accept only odd base-10 positive integers. That is, you should accept `3` and `333433`, but `1110` should be rejected. What has changed from your answer to the previous question?

8. Speaking of threes, construct a FSM which accepts decimal numbers which are divisible by 3. Note that 0, 3, 21, 33960210 are divisible by 3. (**HINT:** You should need exactly 4 states to do this, and think about what it means to be divisible by 3). Try it on some numbers which are divisible by 3 (like 3 and 27) and other numbers which are not divisible by 3 (like 4 and 83).

9. Construct a DFA which accepts all words having the subword `abba` within them. You should check to make sure that `abbba` and `ababa` are rejected and that `abbbabba` is accepted.

10. What is the regular expression which is equivalent to the language accepted by this machine?



11. Construct a FSM over the alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ which accepts words which contain an even number of `a`'s. There are no restrictions on the number of `b`'s or `c`'s. You should verify that `aabaacc`, `b` and `abacababc` are accepted and that `aaacacac` and `abb` are rejected. Hint: You should use two states and keep track of the "parity" (even or odd) of the number of `a`'s you have read in.

12. Find a regular expression for the FSM in Question 11.

13. Construct a FSM over the alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ which accepts words which contain an even number of `a`'s and an odd number of `b`'s. There are no restrictions on the number of `c`'s. You should verify that `aabaacc`, `b` and `abacababc` are accepted and that `aaaacacac` and `bb` are rejected. Hint: You need 4 states, and you should keep track of the parity of both `a`'s and `b`'s.

14. Find a regular expression for the FSM in Question 13. Hint: this is really hard to do by hand. Use JFLAP to do this work for you, which should convince you how hard it is.

15. Construct the FSM for the following regular expression:

$$\mathtt{a}^*\mathtt{bc}^*|\mathtt{a}(\mathtt{bc})^*|(\mathtt{abc})^*$$

(Hint: you should think about creating three distinct FSMs, for each part of the regular expression. Then, connect them using a λ-transition from the start state.)