

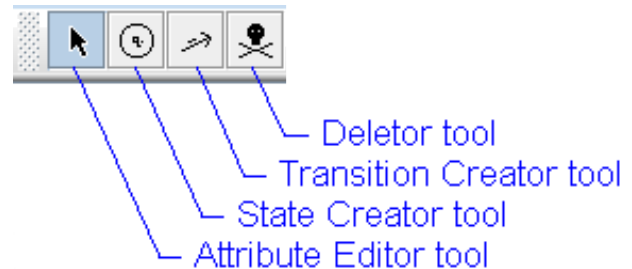
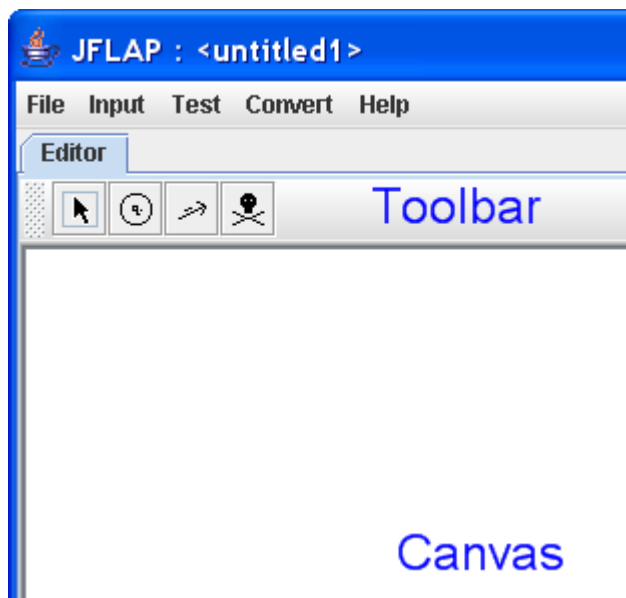
Laboratory – 1

Simulating a DFA from a given regular expression

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



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.



As you can see, the toolbar holds four tools:

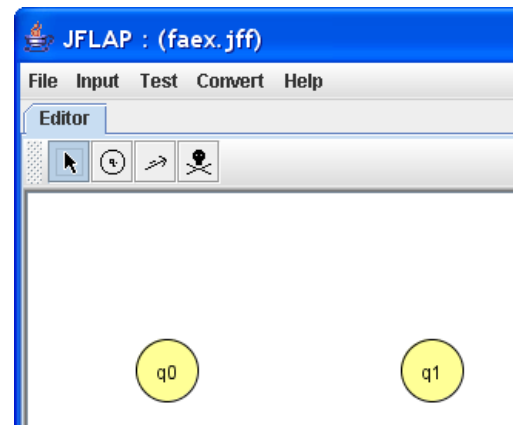
- Attribute Editor tool : sets initial and final states
- State Creator tool : creates states
- Transition Creator tool : creates transitions
- Deleter tool : deletes states and transitions

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.

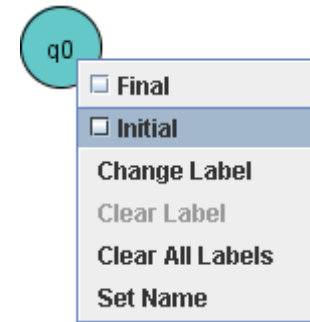
The different modes dictate the way mouse clicks affect the machine. For example, if we are in the State Creator mode, clicking on the canvas will create new states. These modes will be described in more detail shortly.

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



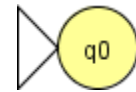
sure how many states we will need, so we created four states. Your editor window should look something like this:



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:

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.

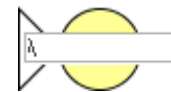



Next, let's create a final state. Arbitrarily, we select q_1 as our final state. To define it as the final state, right-click on the state and click the checkbox **Final**. It will have a double outline, indicating that it is the final state.



Creating Transitions

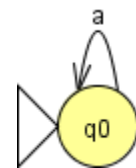
We know strings in our language can start with a 's, so, the initial state must have an outgoing transition on a . We also know that it can start with any number of a 's, which means that the FA should be in the same state after processing input of any number of a 's. Thus, the outgoing transition on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




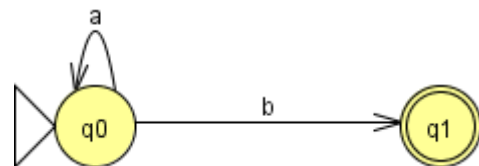
To create such a transition, first select the Transition Creator tool  from the toolbar. Next, click on q_0 on the canvas. A text box should appear over the state:

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:

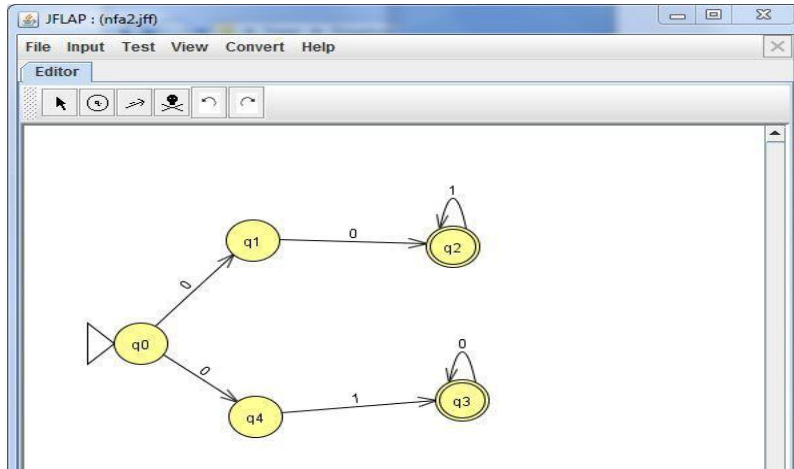


Next, we know that strings in our language must end with a odd number of b 's. Thus, we know that the outgoing transition on b from q_0 must be to a final state, as a string ending with one b should be accepted. To create a transition from our initial state q_0 to our final state q_1 , first ensure that 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.



1. Create a NFA for representing the language $(001^*)|(010^*)$ using JFLAP

Creating DFA and NFA (both combined as FA in JFLAP) is fast and you do not have to set any variables or labels by adding items. Users do not have to set any alphabet before debugging which makes this simulator very straightforward. By double left clicking any transition the user can easily setup every transition.



Assignment

Design NFA, DFA, minimized DFA for the following set of languages:

1. $R = (b|ab^*ab^*)^*$ $\Sigma = \{a, b\}$
2. $R = (ab + a)^*bab$ $\Sigma = \{a, b\}$
3. Set of all strings over $\Sigma = \{0, 1\}$ where each end with '00'
4. Set of all strings over $\Sigma = \{0, 1\}$ having only even no of 0's.
5. Set of all strings over $\Sigma = \{0, 1\}$ having only odd no of 1's.
6. Set of all strings over $\Sigma = \{0, 1\}$ where each string contain substring '0101'
7. Set of all strings over $\Sigma = \{0, 1\}$ where length of each string is at most 5.
8. $R = ((1+0)1)^*100$ $\Sigma = \{0, 1\}$
9. $R = ab(a+ba)^*a^*$ $\Sigma = \{a, b\}$
10. $R = (a|b)^*ab(a|b)^*$ $\Sigma = \{a, b\}$
11. Set of all strings over $\{0, 1\}$ which not start with 10.
12. Set of all strings over $\{0, 1\}$ having odd number of 0's and 1's.
13. Set of all strings over $\{0, 1\}$ having even number of 0's and odd number of 1's.
14. Set of all strings over $\{0, 1\}$ which end with 3 consecutive 1's.
15. Set of all strings over $\{0, 1\}$ which do not have the substring 110.
16. Set of all strings of the form $a^{2n}bb$, $n \geq 0$ over $\{a, b\}$
17. Accept set of all strings start with a followed by any number of b's and ends with a, or set of all strings start with b followed by any number of a's and ends with b.
18. Set of all strings over $\{0, 1\}$ which contain at least 3 a's.

Laboratory – 2

Acceptance of a string from a given regular expression

Regular Expression represents patterns of strings of characters. A regular expression r is completely defined by the set of strings that it matches. This is called the **language generated by the regular expression** and is written as $L(r)$. A regular expression r will also contain characters from the alphabet, but such characters have a different meaning: in regular expression all symbols indicate patterns.

A regular expression is built up out of simpler regular expression using a set of defining rules. Each regular expression r denotes a language $L(r)$. The defining rules specify how $L(r)$ is formed by combining in various ways the language denoted by the sub expression of r .

Finite Automata(FA) is the simplest machine to recognize patterns (regular expression).

A Finite Automata consists of the following :

Q : Finite set of states.
 Σ : set of Input Symbols.
 q : Initial state.
 F : set of Final States.
 δ : Transition Function.

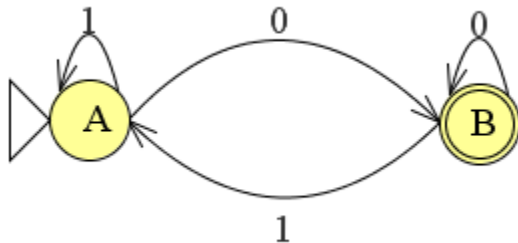
Formal specification of machine is $M = \{ Q, \Sigma, \delta, q, F \}$.

Deterministic Finite Automata (DFA)

DFA consists of 5 tuples $\{Q, \Sigma, q, F, \delta\}$.
 Q : set of all states.
 Σ : set of input symbols. (Symbols which machine takes as input)
 q : Initial state. (Starting state of a machine)
 F : set of final state.
 δ : Transition Function, defined as $\delta : Q \times \Sigma \rightarrow Q$.

In a DFA, for a particular input character, the machine goes to one state only. A transition function is defined on every state for every input symbol. Also in DFA null (or ϵ) move is not allowed, i.e., DFA cannot change state without any input character.

For example, below DFA with $\Sigma = \{0, 1\}$ accepts all strings ending with 0.



For this DFA input and output will be as follows-

Input	Result
ϵ	Reject
0 1 0	Accept
0 0 1 1	Reject
1 0 1 1	Reject
0	Accept
1	Reject
0 0 0 0	Accept
0 1 0 1 0	Accept

```
#include <stdio.h>
#include <string.h>
//starting state
int dfa = 1;

// This function is for
// the starting state A of DFA
void A(char c)
{
    if (c == '0')
        dfa = 2;
    else if (c == '1')
        dfa = 1;
    // -1 is used to check for any invalid
    symbol
    else
        dfa = -1;
}
```

```
// This function is for the first state B of DFA
void B(char c)
{
    if (c == '0')
        dfa = 2;
    else if (c == '1')
        dfa = 1;
    else
        dfa = -1;
}
```

```
int isAccepted(char str[])
{
```

```
// store length of string
int i, len = strlen(str);

for (i = 0; i < len; i++) {
    if (dfa == 1)
        A(str[i]);

    else if (dfa == 2)
        B(str[i]);

    else
        return 0;
}
if (dfa == 2)
    return 1;
else
    return 0;
}
```

```
int main()
{
    char str[50];
    printf("enter the string");
    gets(str);
    if (isAccepted(str))
        printf("%s is ACCEPTED",str);
    else
        printf("%s is NOT
ACCEPTED",str);
    return 0;
}
```

Assignment

Check the acceptance of the string for the following set of languages:

1. $R=(b|ab^*ab^*)^* \Sigma=\{a,b\}$
2. $R=(ab+a)^*bab \Sigma=\{a,b\}$
3. Set of all strings over $\Sigma=\{0,1\}$ where each end with '00'
4. Set of all strings over $\Sigma=\{0,1\}$ having only even no of 0's.
5. Set of all strings over $\Sigma=\{0,1\}$ having only odd no of 1's.
6. Set of all strings over $\Sigma=\{0,1\}$ where each string contain substring '0101'
7. Set of all strings over $\Sigma=\{0,1\}$ where length of each string is at most 5.
8. $R=((1+0)1)^*100 \Sigma=\{0,1\}$
9. $R=ab(a+ba)^*a^* \Sigma=\{a,b\}$
10. $R=(a|b)^*ab(a|b)^* \Sigma=\{a,b\}$
11. Set of all strings over $\{0, 1\}$ which not start with 10.
12. Set of all strings over $\{0, 1\}$ having odd number of 0's and 1's.
13. Set of all strings over $\{0, 1\}$ having even number of 0's and odd number of 1's.
14. Set of all strings over $\{0, 1\}$ which end with 3 consecutive 1's.
15. Set of all strings over $\{0, 1\}$ which do not have the substring 110.
16. Set of all strings of the form $a^{2n}bb$, $n \geq 0$ over $\{a, b\}$
17. Accept set of all strings start with a followed by any number of b's and ends with a, or set of all strings start with b followed by any number of a's and ends with b.
18. Set of all strings over $\{0, 1\}$ which contain at least 3 a's.