

# Mealy Machines

## Contents

### [Definition](#)

### [Differences between a Mealy Machine and an FA](#)

- [No Final State](#)
- [Transition Output](#)
- [Producing Output from an Input String](#)
- [No Nondeterminism](#)

### [Proceed to Mealy Machine Examples](#)

## Definition

JFLAP defines a Mealy machine  $M$  as the sextuple  $M = (Q, \Sigma, \Gamma, \delta, \omega, q_s)$  where

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

$\Sigma$  is the finite input alphabet

$\Gamma$  is the finite output alphabet

$\delta$  is the transition function,  $\delta : Q \times \Sigma \rightarrow Q$

$\omega$  denotes the output function,  $\omega : Q \times \Sigma \rightarrow \Gamma$

$q_s$  (is a member of  $Q$ ) is the initial state

Mealy machines are different than [Moore machines](#) in the output function,  $\omega$ . In a Mealy machine, output is produced by its transitions, while in a Moore machine, output is produced by its states.

To start a new Mealy machine, select the **Mealy Machine** option from the main menu.



Starting a new Mealy machine


## Differences between a Mealy Machine and an FA

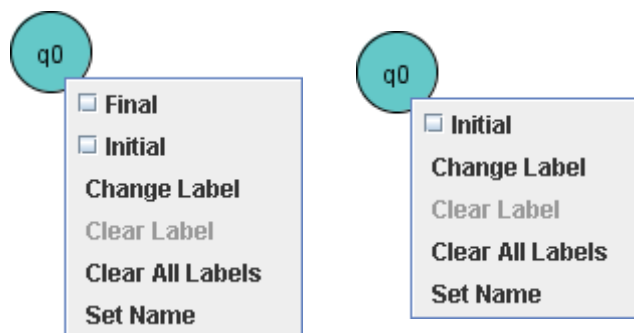
A Mealy machine is very similar to a [Finite Automaton](#) (FA), with a few key differences:

- It has [no final states](#).
- Its [transitions produce output](#).
- It does not accept or reject input, instead, it [generates output from input](#).
- Lastly, Mealy machines [cannot have nondeterministic states](#).

Let's go through these points.

### No Final State

In an FA, when you have the Attribute Editor tool selected (you may do so by clicking the  button), right-clicking on a state it will produce a pop-up menu that allows you to, among other things, set it to be a final state. In a Mealy machine, that option is not available.



An FA state menu

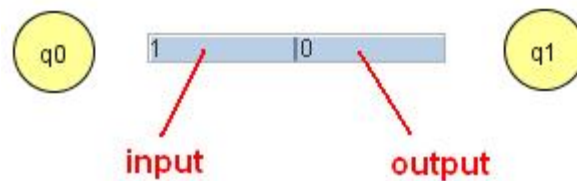
A Mealy machine state menu

A Mealy machine does not have final states because it does not accept or reject input. Instead, each transition produces output, which will be described below.

## Transition Output

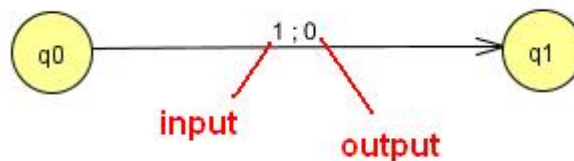
A Mealy machine produces output each time it takes a transition.

Creating a Mealy machine is the same as creating an FA with the exception of creating its transitions. In a Mealy machine, each transition produces output. When you are creating a transition, two blanks appear instead of one. The first blank is for the input symbol, the second blank is for the output symbol.



Creating a transition

When the transition is created, its label will be two symbols separated by a semicolon, ";". The input symbol is to the left of the semicolon, and the output symbol is to its right.



Transition created

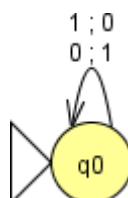
Thus, when in  $q_0$  with input of "1", the machine will take the transition to  $q_1$  and produce the output "0".

With each transition producing output, the Mealy machine can produce output from an input string.

## Producing Output from an Input String

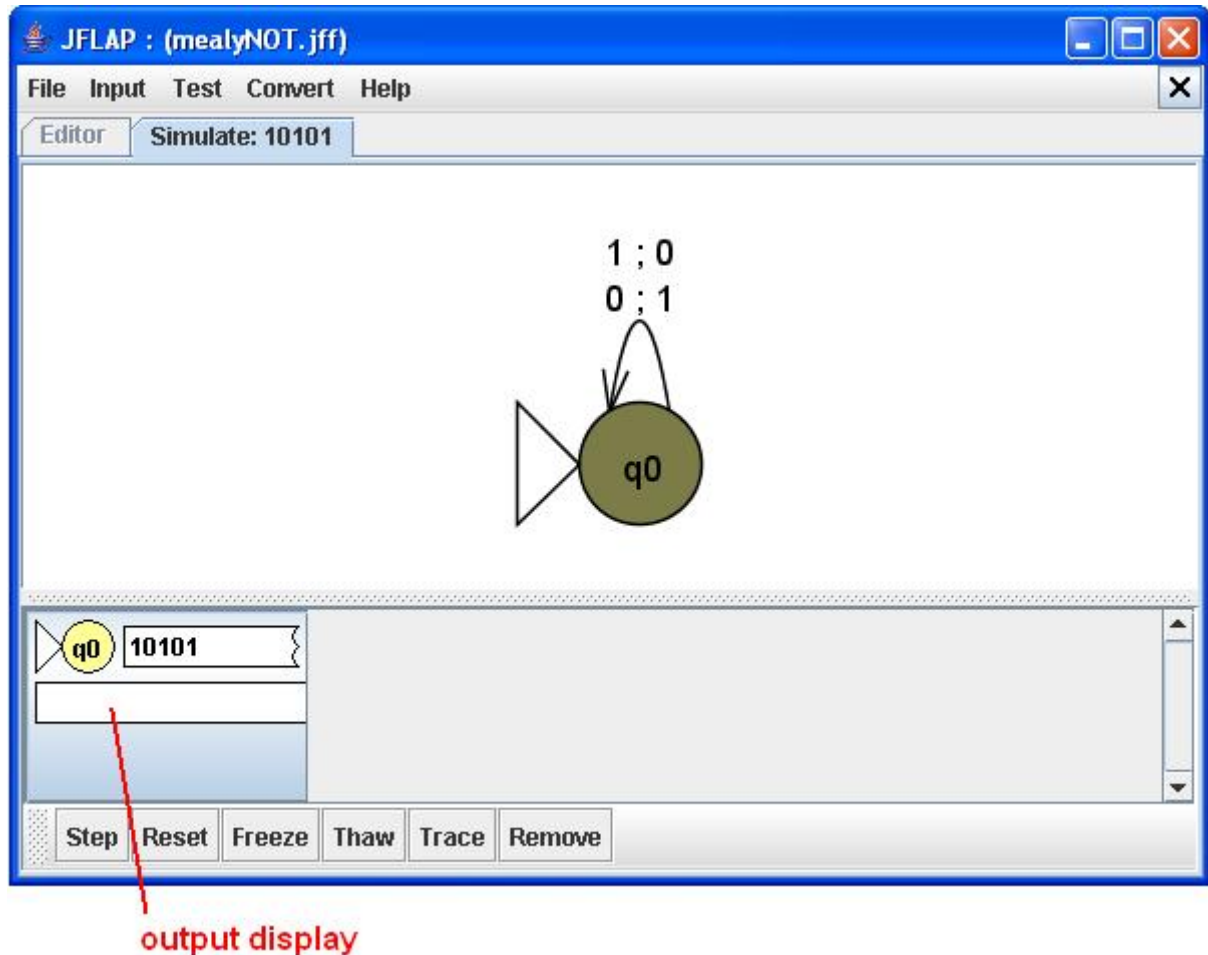
Instead of accepting or rejecting input, a Mealy machine produces output from an input string.

Let's look at this simple Mealy machine, which can be downloaded through [mealyNOT.jff](#):



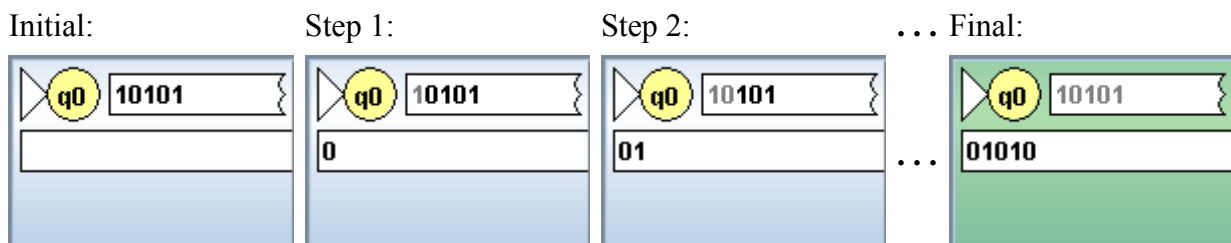
## A simple Mealy machine

When we select the menu option **Input : Step...** and enter your input, we get a display similar to that of an FA, except with another piece of tape below the input tape. This displays the output of the machine at the current step.

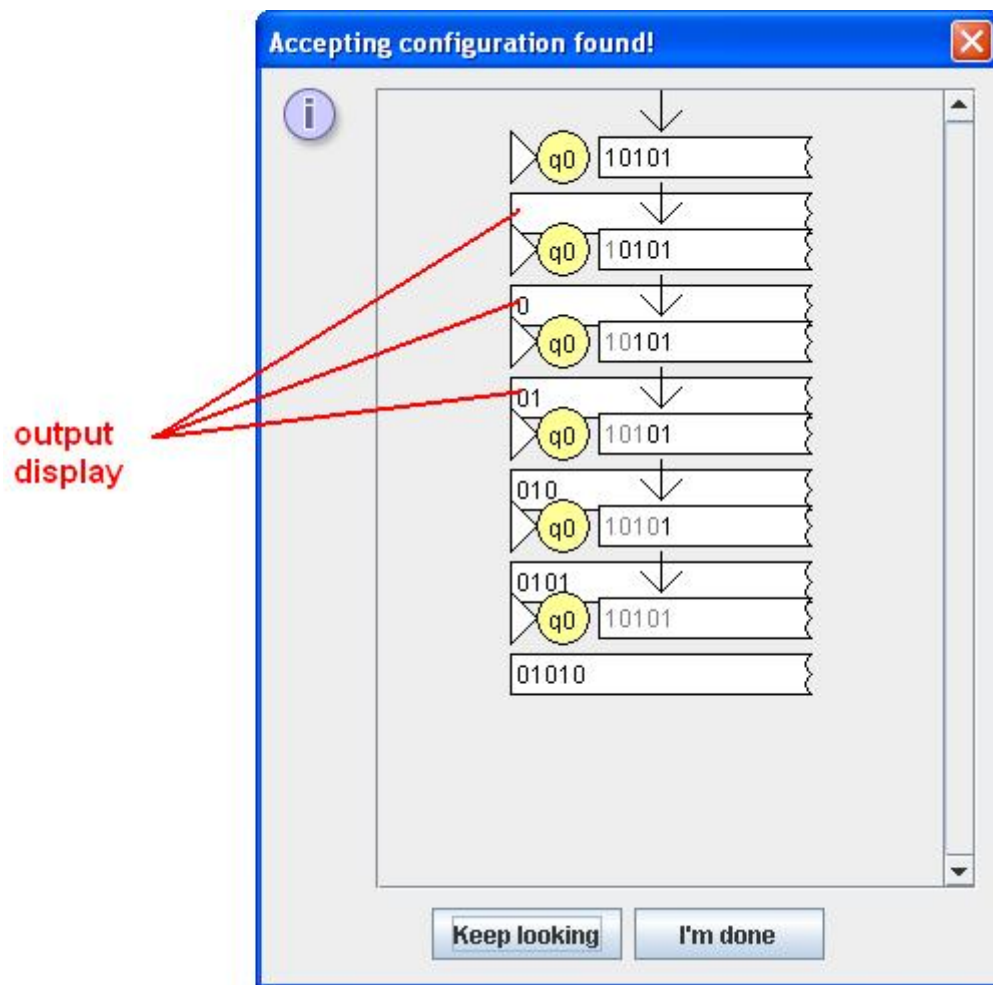


The output display

At each step. The output tape updates itself to display the total output that has been produced so far:

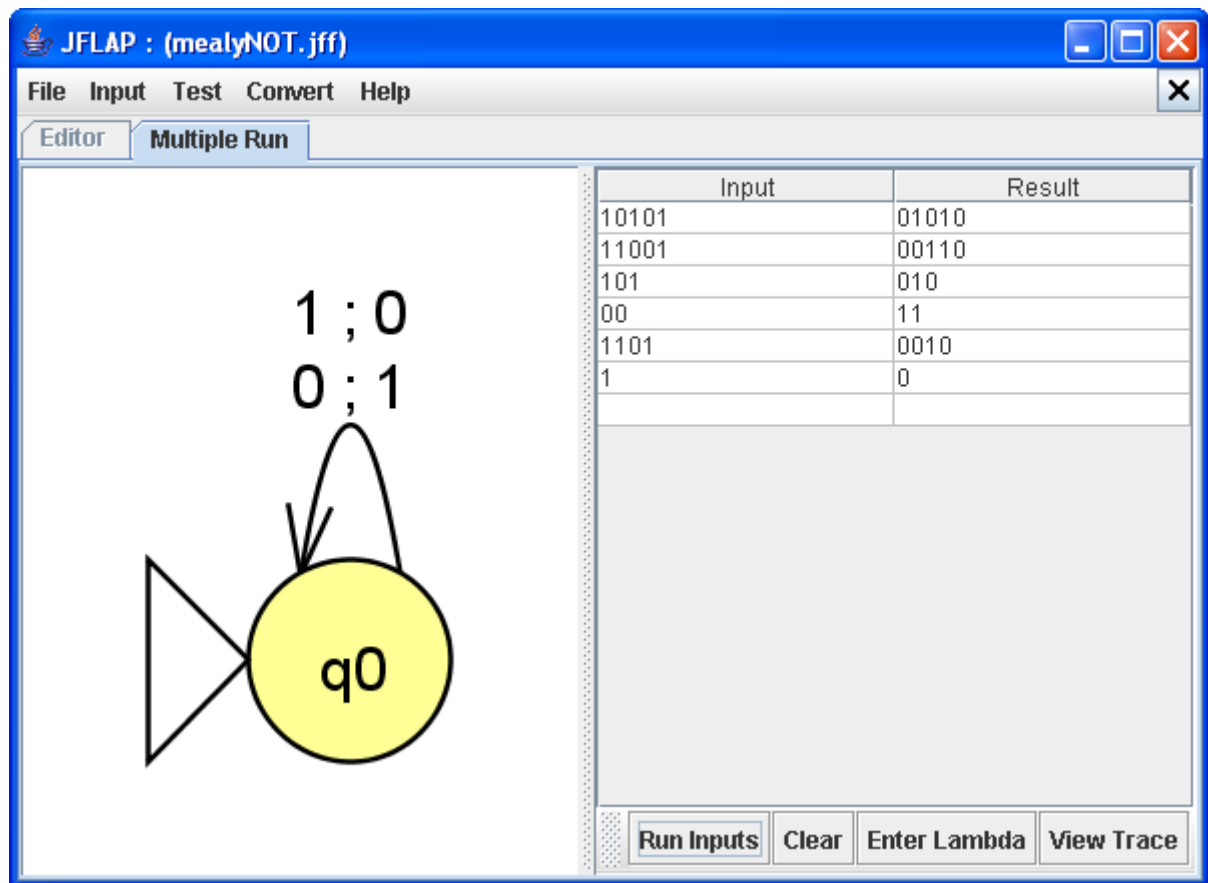


The menu option **Input : Fast Run...** works similarly, with the output being displayed in a tape under the input tape:



Fast run output display

Similarly, when we select **Input : Multiple Run**, output is displayed in the **Result** column.

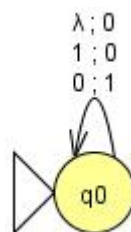


Multiple run output display

## No Nondeterminism

As Mealy machines map an input to a unique output, nondeterminism cannot exist in a Mealy machine. Although we still will be able to build a Mealy machine that has nondeterminism, we will not be able to run input on it.

For instance, let's modify our machine slightly to make:



Mealy machine with nondeterminism

We we try to run it on an input, with **Input : Step...**, **Input : Fast Run...**, or **Input : Multiple Run...**, we will get an error message asking us to remove the nondeterministic states:



Select **Test : Highlight Nondeterminism** to view the nondeterministic states. Remove the nondeterminism in order to be able to run input on the Mealy machine.

**This concludes our brief tutorial on Mealy machines.**

## Mealy Machine Examples

### Contents

[Back to Mealy Machines](#)

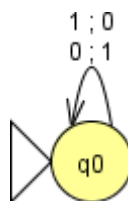
[Example 1: NOT](#)

[Example 2: Vending Machine](#)

### Example 1: NOT

Let's start with a simple Mealy machine that takes an input bit string  $b$  and produces the output  $\text{NOT}(b)$ .

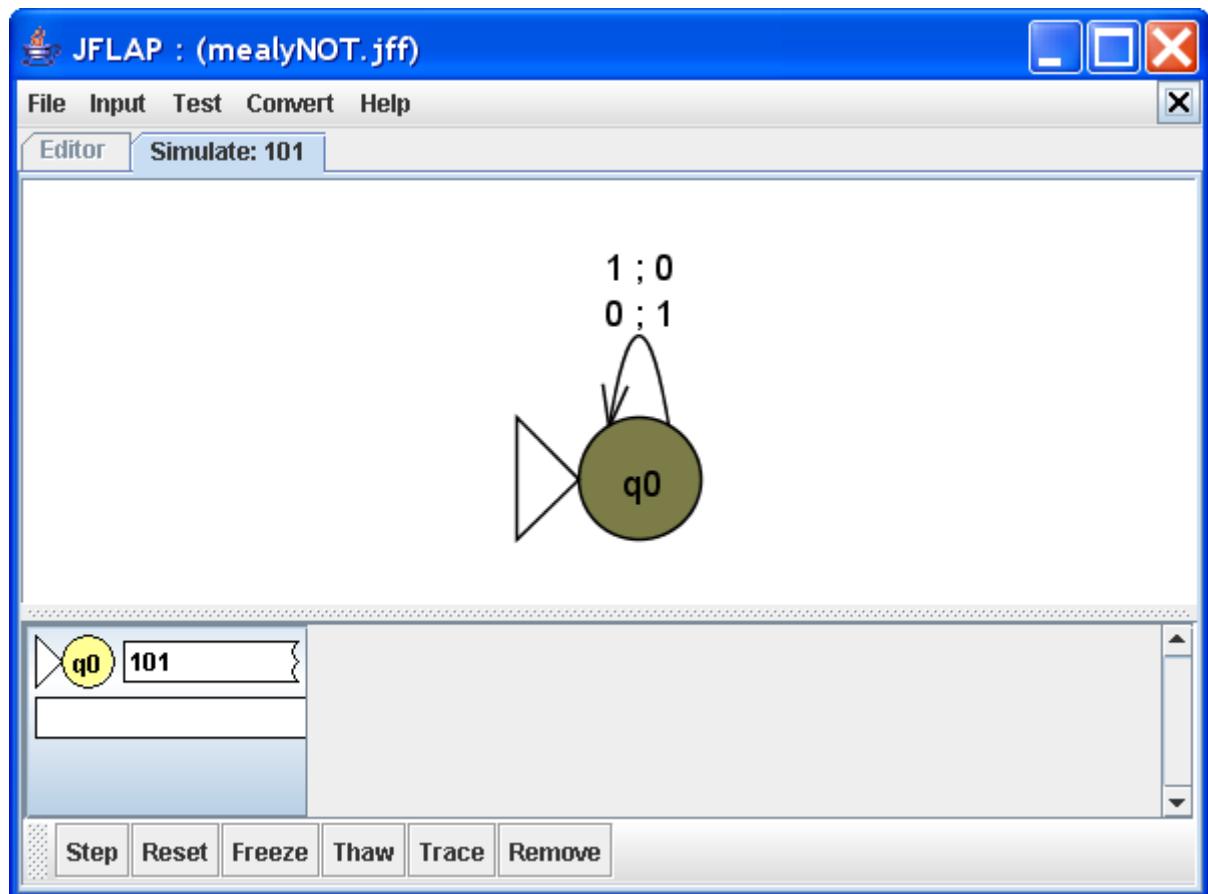
The machine should look like this, and can be downloaded through [mealyNOT.jff](#):



A Mealy machine that produces  $\text{NOT}(b)$

As you can see, the initial state has two outgoing transitions, one for the input "0" that produces the output "1", and one for the input "1" that produces the output "0". Thus, for every bit  $b_i$  of the input, this machine produces the output  $\text{NOT}(b_i)$ , and takes the transition back to the same state so it may similarly process the next bit of the input.

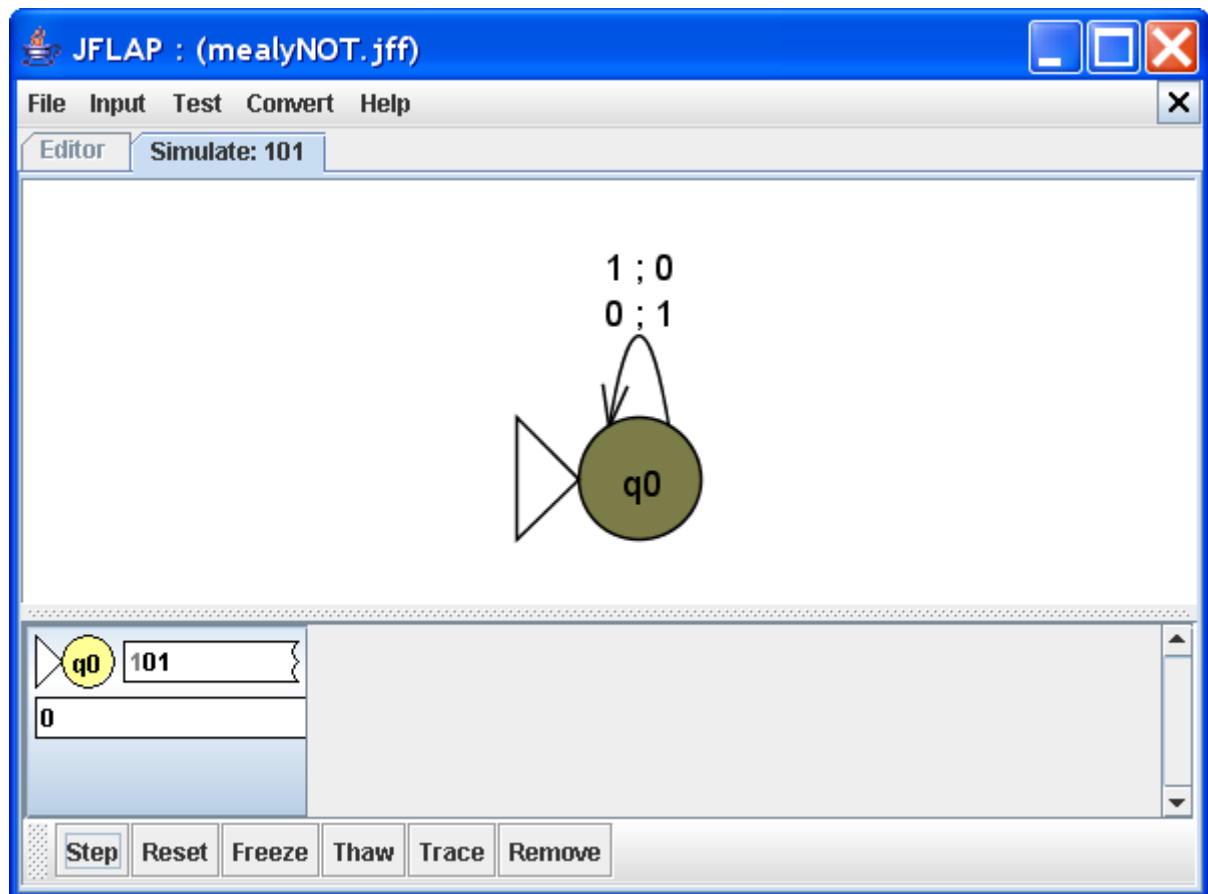
Let's step the machine through the input "101".



Initial set up

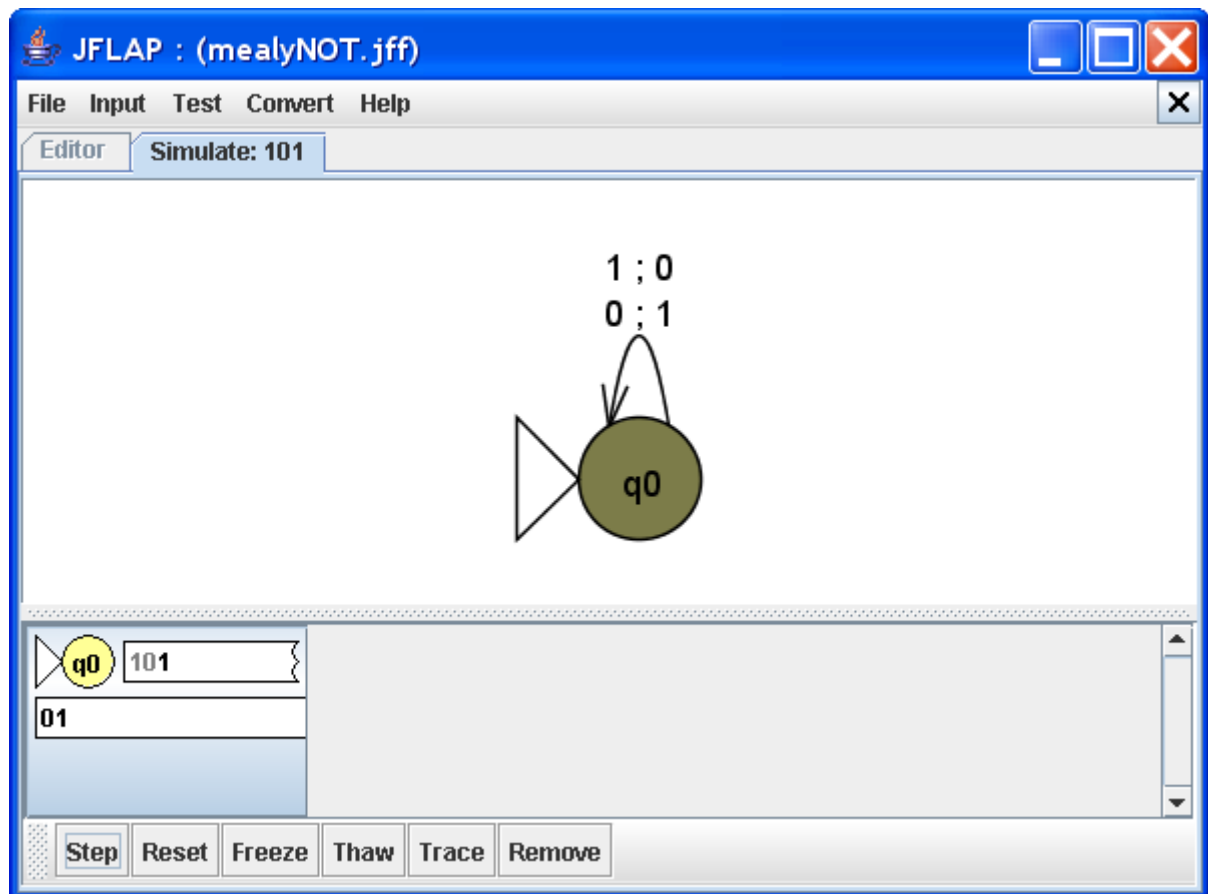
The machine is in its initial, and only, state,  $q_0$ . The two outgoing transitions from  $q_0$  each produce the negation of the input bit. Click **Step** to process the next bit of the input.





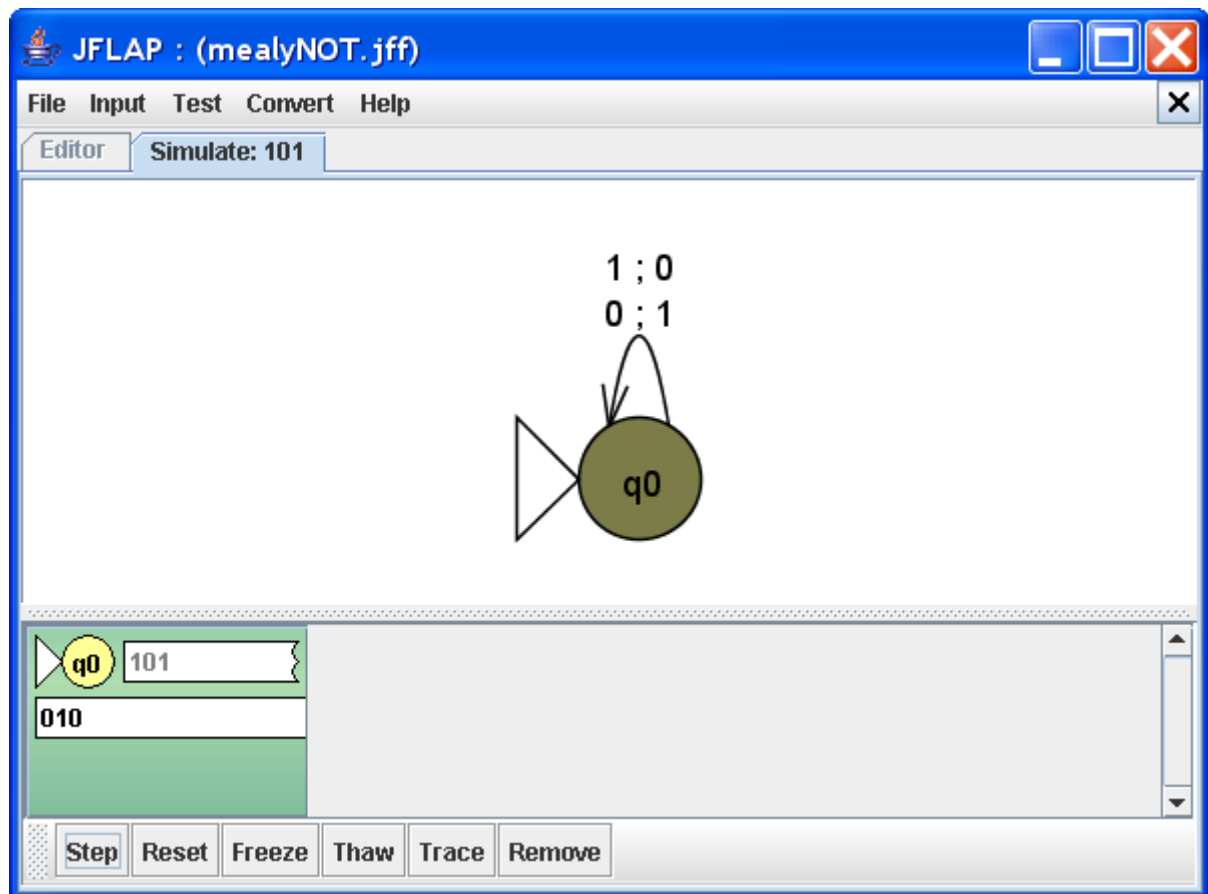
### Step 1

Upon processing the first bit of the input, "1", the machine takes the transition  $1 ; 0$ , producing the output "0", which is the negation of the input bit. This transition takes the machine back to its only state,  $q_0$ . Click **Step** to process the next bit of input.



## Step 2

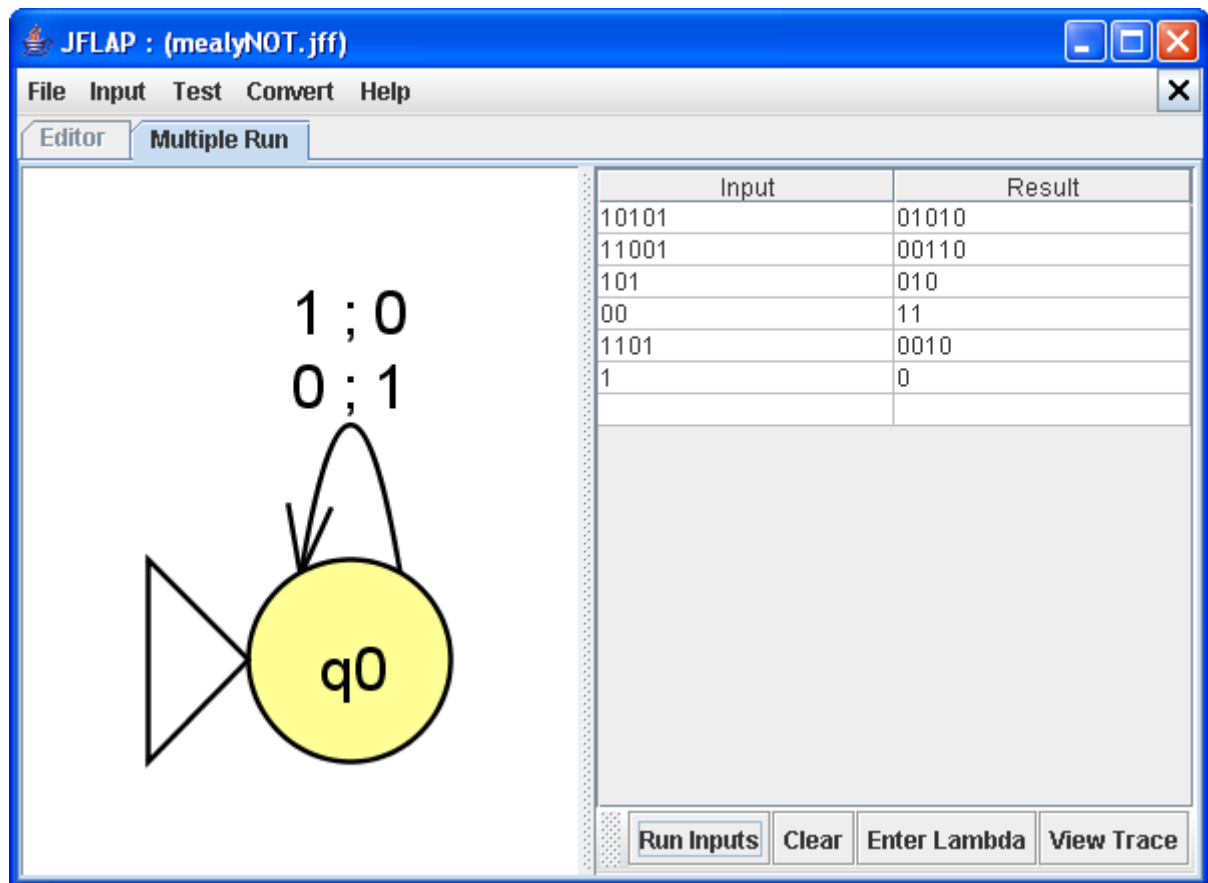
Upon processing the next bit of the input, "0", the machine takes the transition  $0 ; 1$ , producing the output "1", which is the negation of the input bit. This transition takes the machine back to  $q_0$ . We have seen that state  $q_0$  properly processes both input bit "0" and "1", and returns the machine to  $q_0$ , showing that the machine will properly process any input bit. Click **Step** to complete the process.



Step 3

As we can see, the machine indeed produces the negation of its input bit string.

We can also test this machine on multiple strings.



Testing the machine on multiple strings

As you can see, it produces  $\text{NOT}(b)$  in every instance.

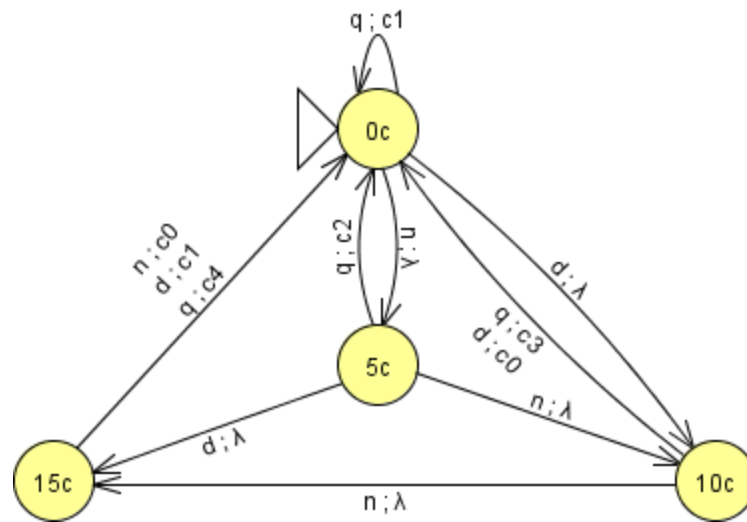
You can download the file [here](#) to try it out.

## Example 2: Vending Machine

(Example adapted from *Theory of Finite Automata with an Introduction to Formal Languages* by John Carroll and Darrell Long.)

Next, let's consider a more complex machine. We shall build a vending machine that dispenses one piece of candy once enough money has been inserted, and returns the appropriate amount of change. (To see a completed version, feel free to download [mealyVending.jff](#).)

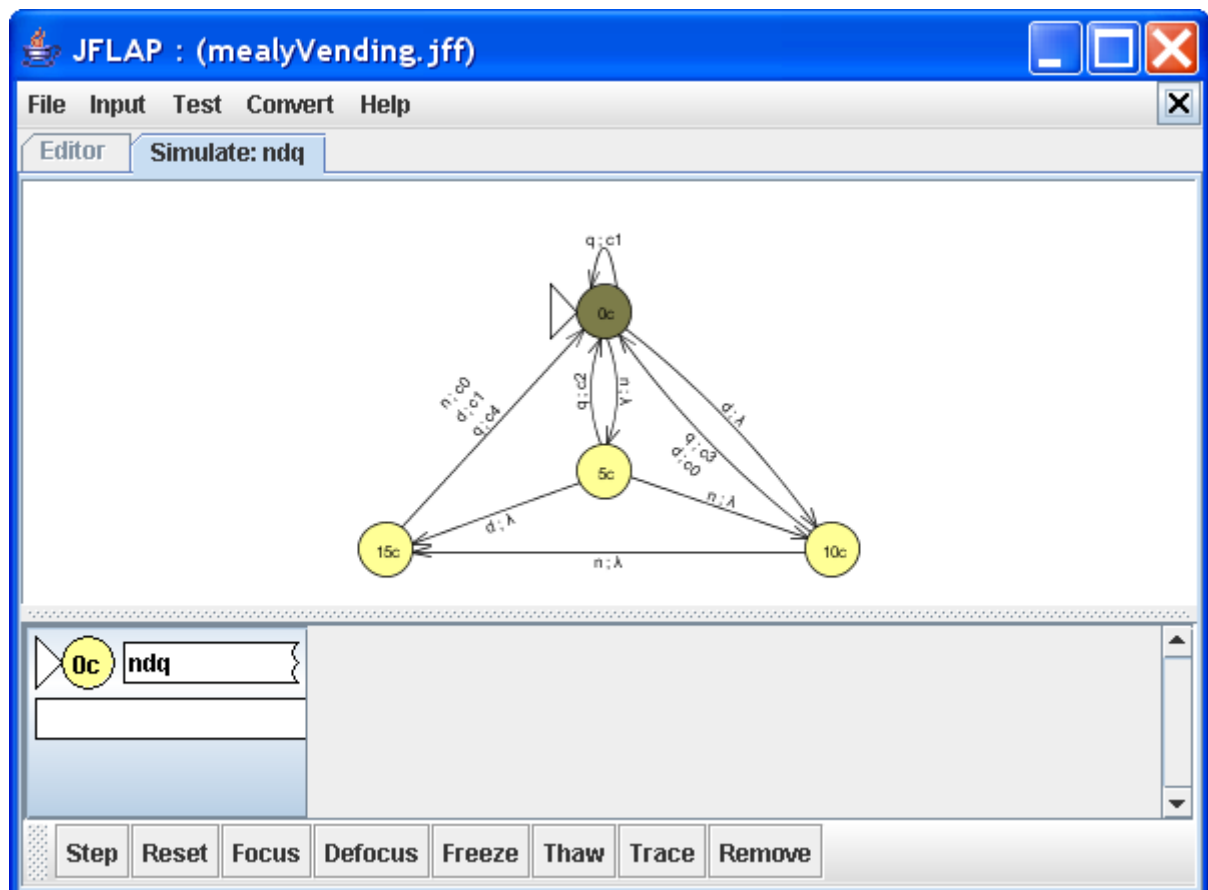
Let  $V = (\{0c, 5c, 10c, 15c\}, \{n, d, q\}, \{c_0, c_1, c_2, c_3, c_4\}, \delta, \omega, 0c)$ . (See the Mealy machine [definition](#).)  $V$  describes a vending machine that dispenses 20¢ candy bars.



Mealy machine V

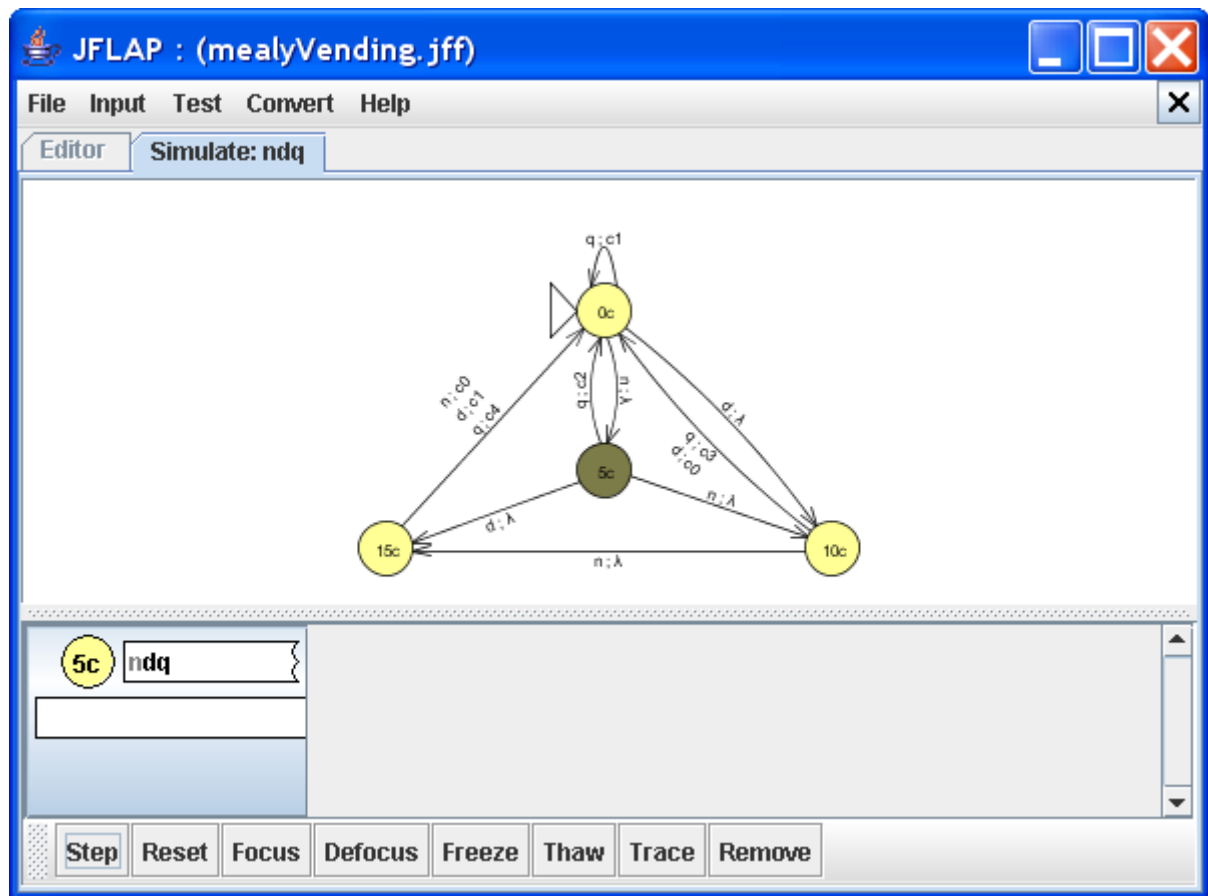
This Mealy machine has four states,  $0c$ ,  $5c$ ,  $10c$ , and  $15c$ , each describing the amount of money inserted in the vending machine,  $0\text{¢}$ ,  $5\text{¢}$ ,  $10\text{¢}$ , and  $15\text{¢}$ . Its input alphabet of  $n$ ,  $d$ , and  $q$ , denotes the insertion of nickels, dimes, and quarters respectively. In its output alphabet,  $\lambda$  denotes the vending machine doing nothing, while  $c_0$ ,  $c_1$ ,  $c_2$ ,  $c_3$ , and  $c_4$ , denote the machine dispensing candy and no, one, two, three and four nickels respectively as change.

Let's step through the input  $ndq$ . (That is, someone inserted a nickel, a dime, followed by a quarter.)



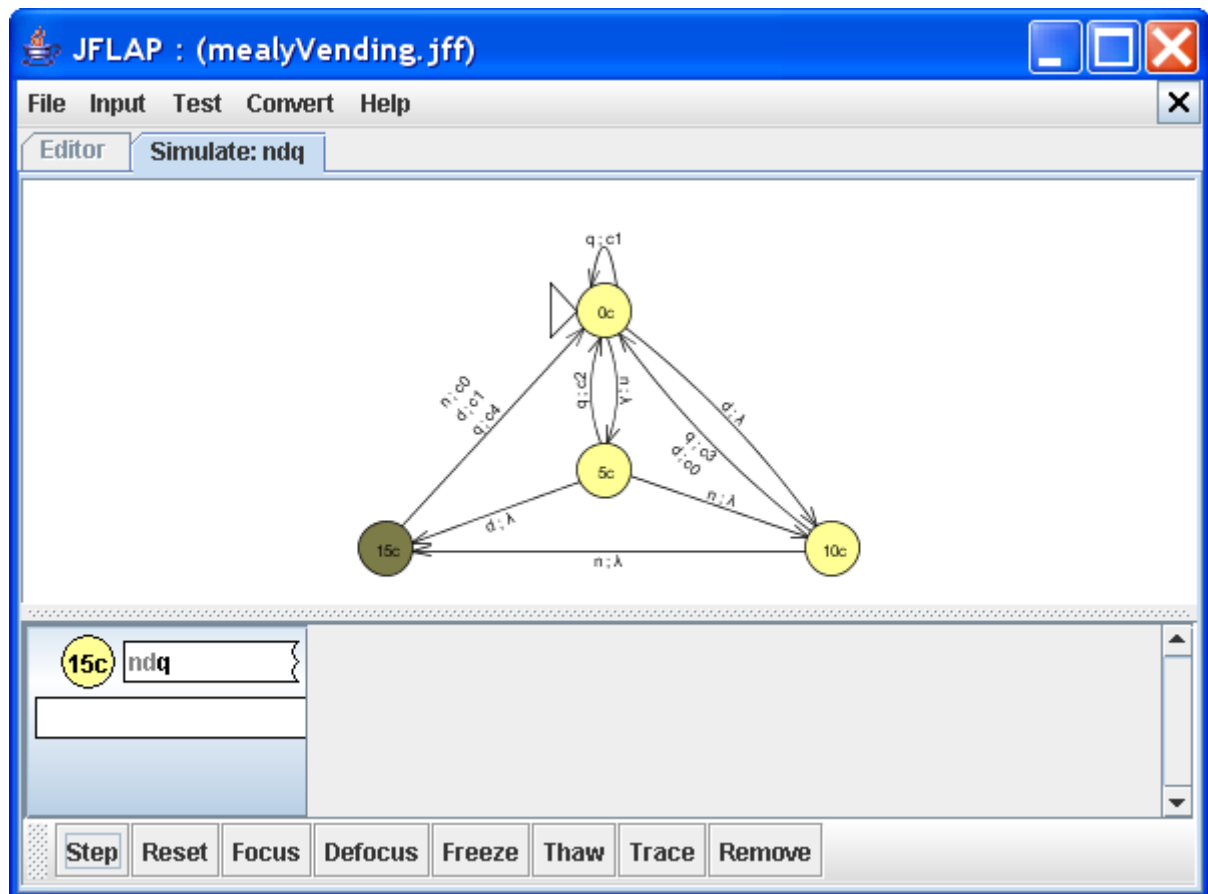
No coins inserted

The machine starts at  $0c$ , as it has not had any money inserted yet. Click **Step** to insert the first nickel.



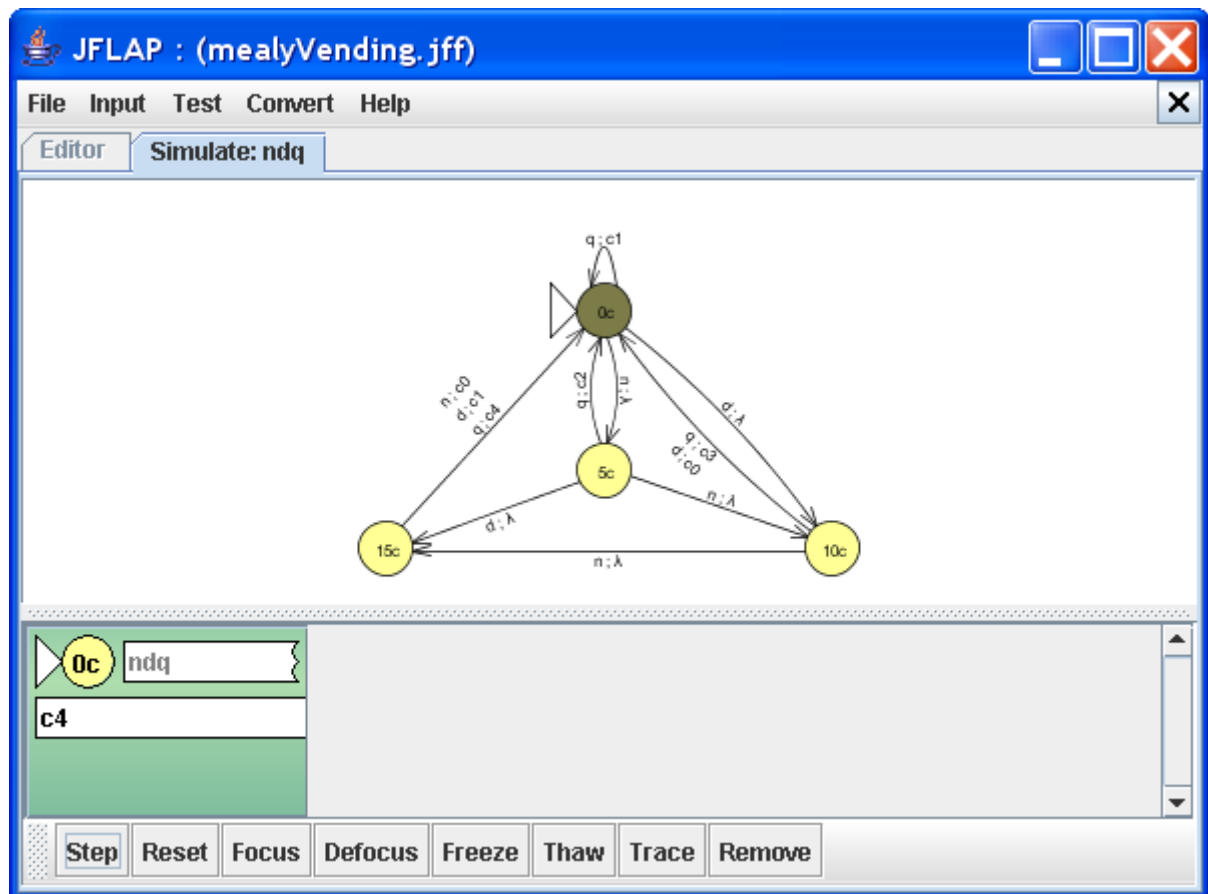
First coin (a nickel) inserted

The machine takes the transition  $n ; \lambda$  from  $0c$  to  $5c$ , denoting that  $5\phi$  has been inserted. The transition produces no output as not enough money has been inserted. Click **Step** to insert the next coin, a dime.



Second coin (a dime) inserted

The machine takes the transition  $d ; \lambda$  from 5c to 15c, indicating that 15¢ has been accumulated. Again, the transition produces no output as not enough money has been inserted. Click **Step** to insert the last coin, a quarter.

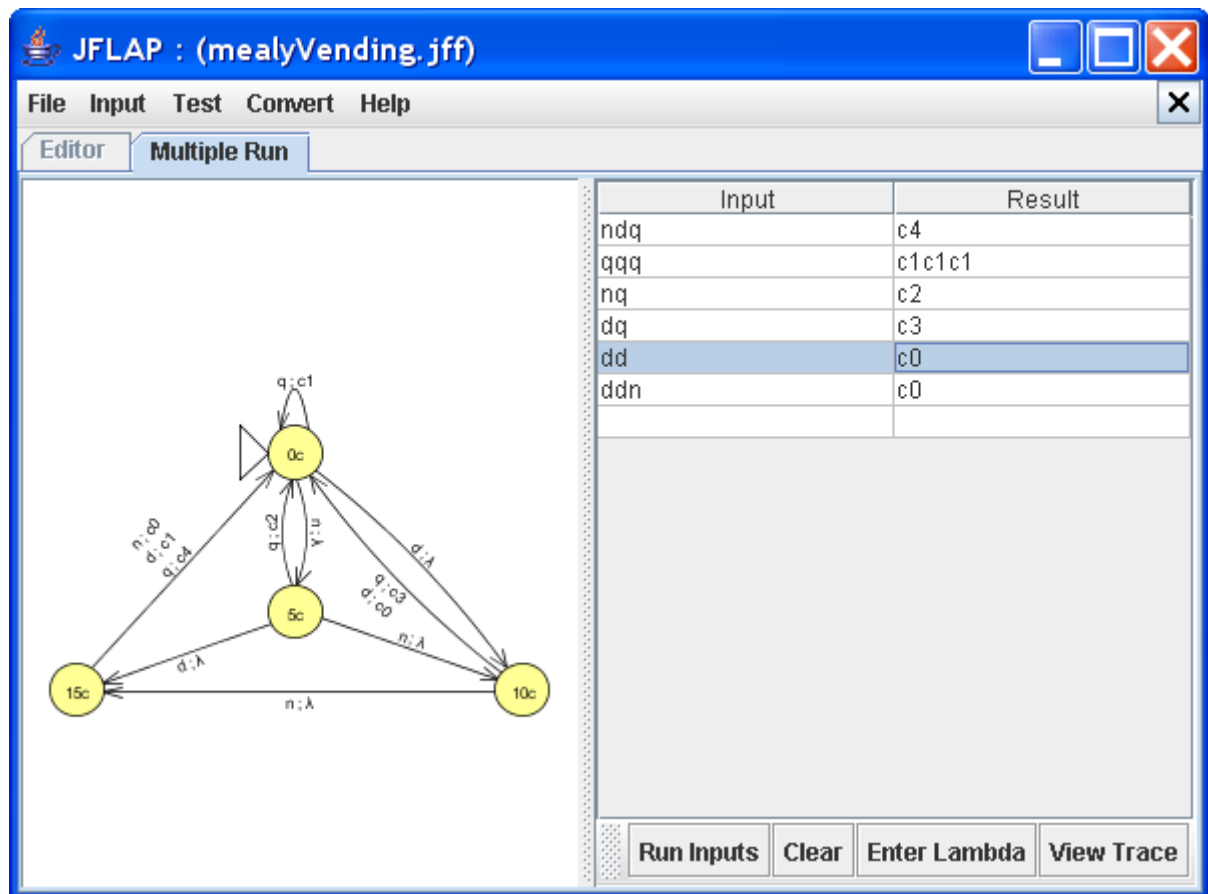


Last coin (a quarter) inserted

The machine takes the transition  $q; c_4$  from  $15c$  to  $0c$ , back to the initial state. The transition produces output  $c_4$ , and the vending machine dispenses a piece of candy and four nickels. Thus, once it has accumulated sufficient money, it dispenses a piece of candy and the correct amount of change, just as we have [described](#).

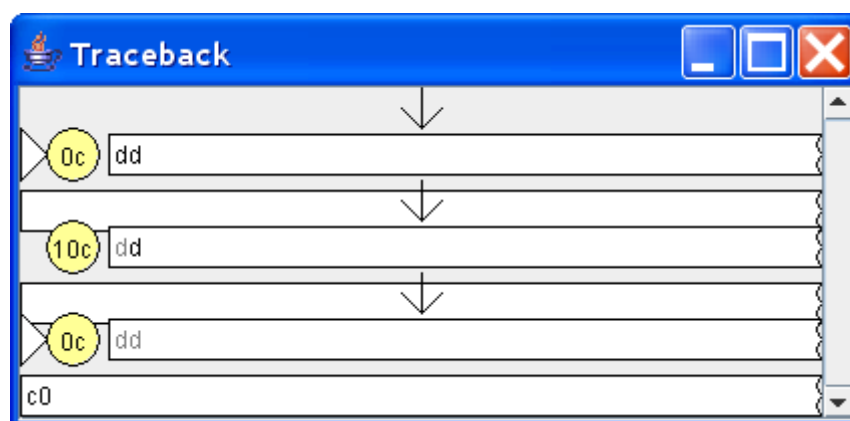
Let's try our vending machine out on different sets of change.





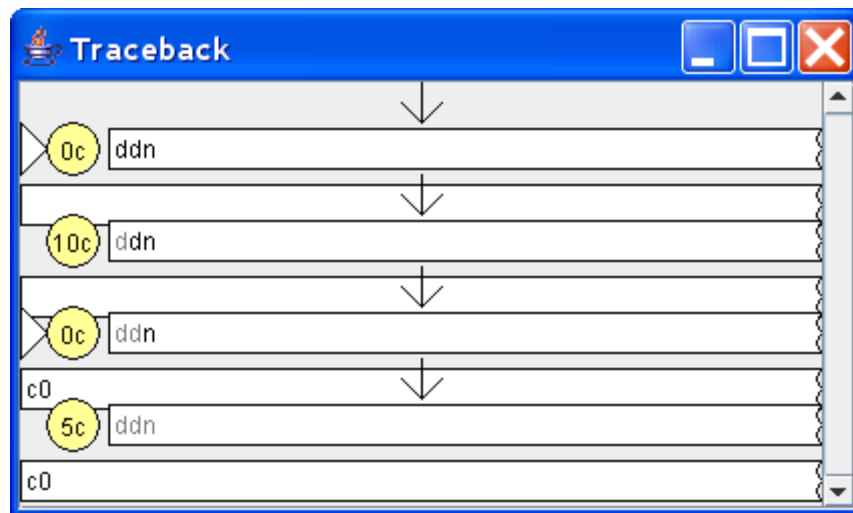
Running the vending machine on different sets of coins

As you can see, each time our vending machine accumulates sufficient money, it dispenses a piece of candy and the appropriate amount of change. The last two inputs, *dd* and *ddn*, although different, produce the same output  $c_0$ . This is because they end in different states. Effectively, for input *ddn* the vending machine still has 5¢ in it.



The trace for input *dd*

For input *dd*, the state we end in is 0c, indicating there is no extra change in the vending machine.



The trace for input *ddn*

For input *ddn*, the state we end in is *5c*, indicating that we still have 5¢ of change in it.