Theory of Automata
Finite Automata with Output

Hafiz Tayyeb Javed

Week 7 Lecture 01

Week 7 Lecture 02

Contents

- Moore Machines
- Mealy Machines
- Moore = Mealy

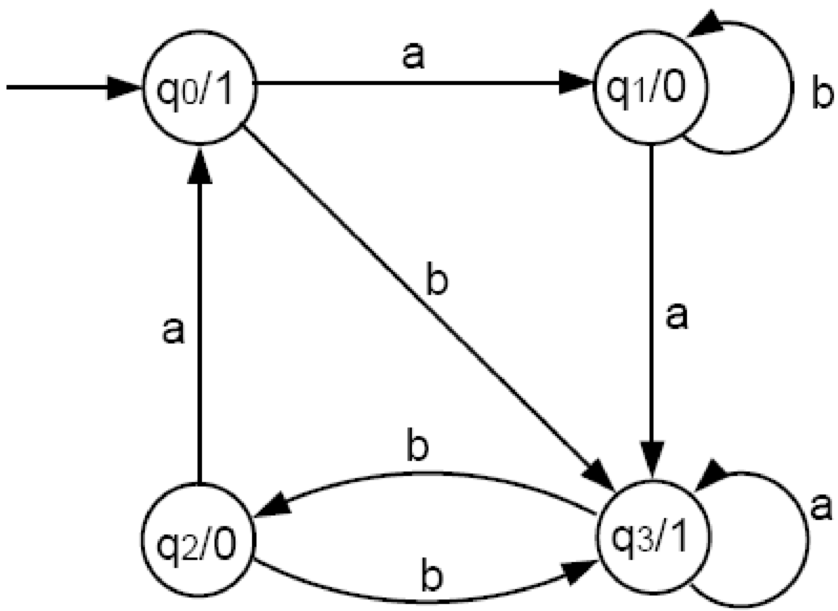
Both Machines are not for "Accepting" or "Rejecting" the Language but only "recognize" 'trace' the language or pattern

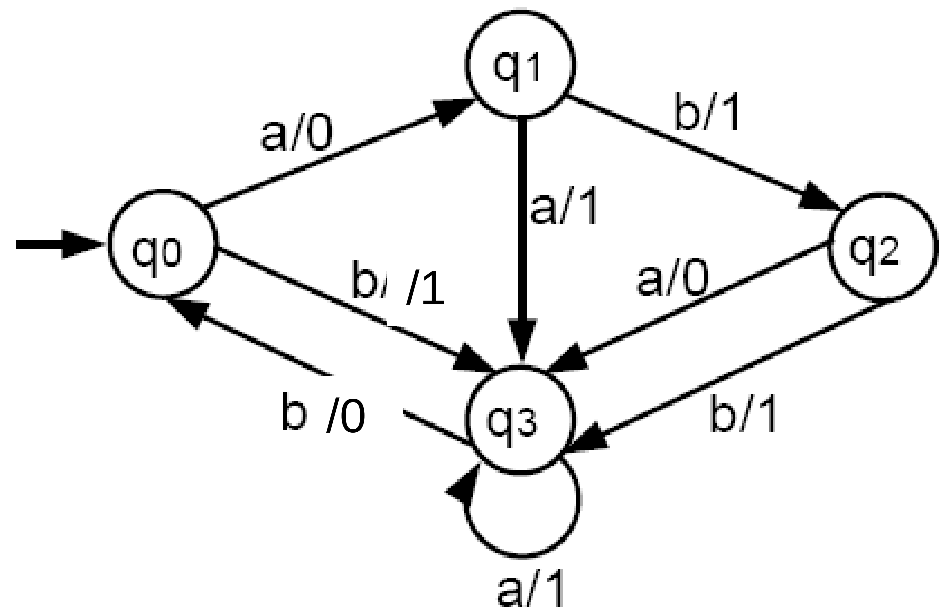
Benefits

- What is we want to do following to the word
  - count the occurrence of a certain substring
  - get the incremented number
  - mark the locations of the substring in the word
  - take the complement of the number
  - Get the parity of the number

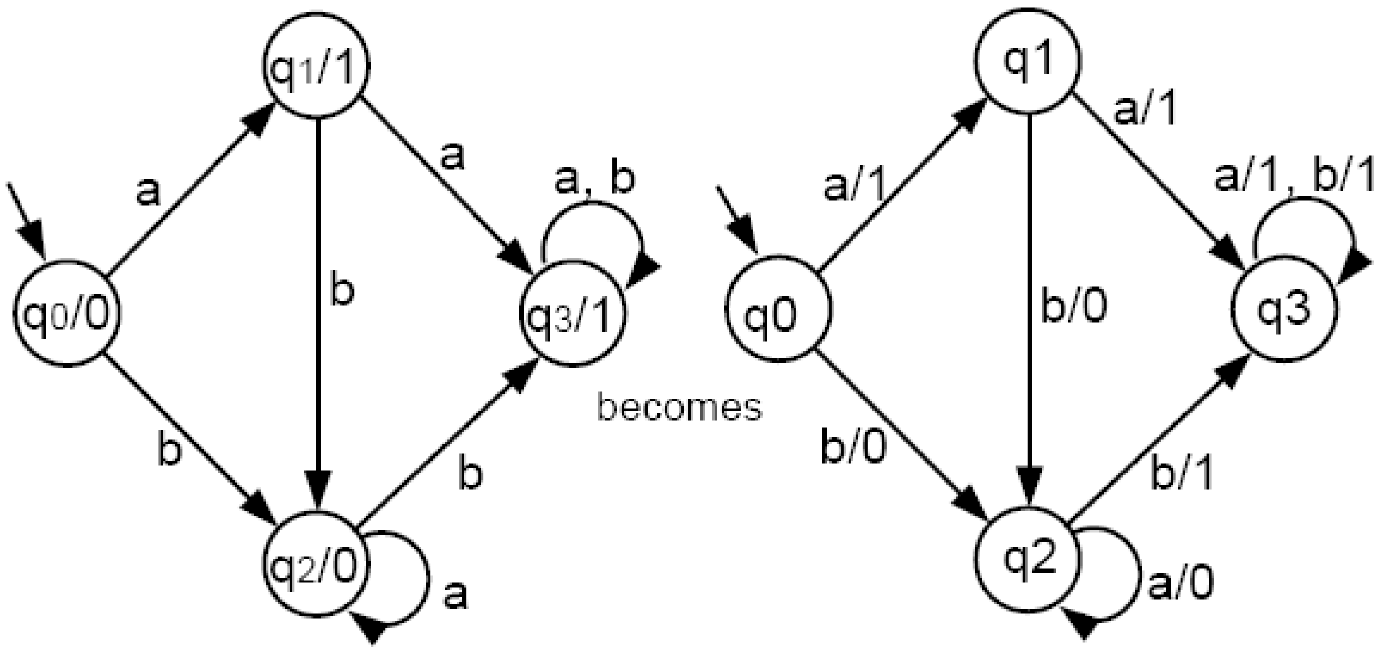
- *Final accepted/rejected may not be required*
- *Not limited by the size of the buffer if we want to wait for the final output*

Moore vs Mealy Machines

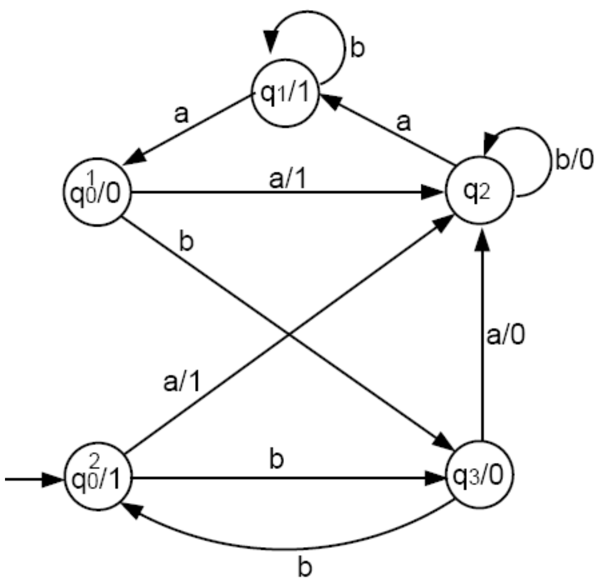


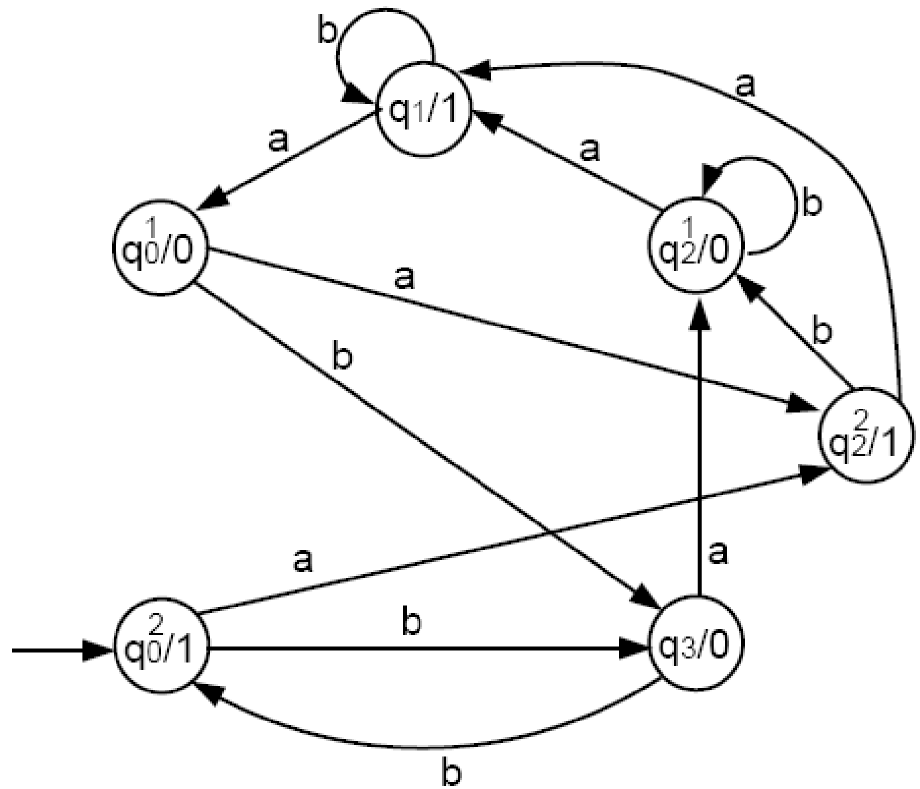


Moore vs Mealy Machines



Moore vs Mealy Machines





Moore Machine Definition

Moore machine is a collection of five things:

1. A finite set of states q_0, q_1, q_2, ..., where q_0 is designated as the start state.
2. An alphabet of letters for forming the input string = {a, b, c, …}.
3. An alphabet of possible output characters **Γ** = {0, 1, 2, …}.
4. A transition table that shows for each state and each input letter what state is reached next.
5. An output table that shows what character from **Γ** is printed by each state as it is entered.

Notes

- We did not assume that the input alphabet is the same as the output alphabet **Γ**.
- To keep the output alphabet separate from the input alphabet, we give it a different name **Γ** (instead of ∑) and use number symbols {0, 1, …} (instead of {a, b, …}).
- We refer to input symbols as **letters**, whereas we refer to output symbols as **characters**.
- We adopt the policy that a Moore machine always begins by printing the character dictated by the mandatory start state. So, if the input string has 7 letters, then the output string will have 8 characters, because it includes 8 states in its path.

Notes Contd.

- A Moore machine does not define a language of accepted words, because there is no such thing as a final state.

- Every possible input string creates an output string. The processing is terminated when the last input letter is read and the last output character is printed.

- There are some subtle ways to turn Moore machines into language definers.

Example: Moore machine defined by a table

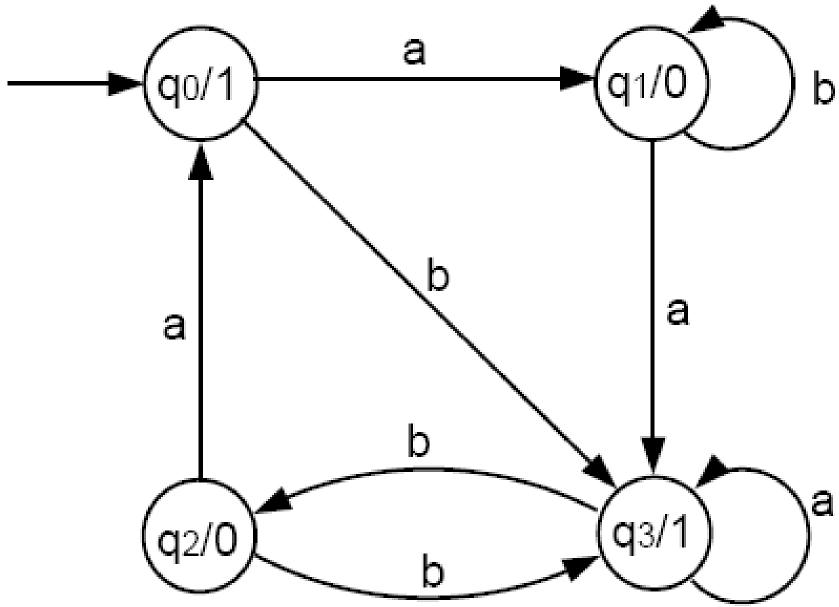
- Input alphabet: $\Sigma = \{a, b\}$
- Output alphabet: $\Gamma = \{0, 1\}$
- Names of states: q_0, q_1, q_2, q_3 with q_0 being the start state.
- Transition and output table (combined):

Old State	Output by Old State	New state after *a*	New state after *b*
q_0	1	q_1	q_3
q_1	0	q_3	q_1
q_2	0	q_0	q_3
q_3	1	q_1	q_2

Pictorial Representation

- Moore machines have pictorial representations similar to FAs.
- The difference is that inside each state, in addition to the state name, we also specify the output character printed by that state, using the format *state – name/output*.
- Hence, the Moore machine in the above example has the following pictorial representation:

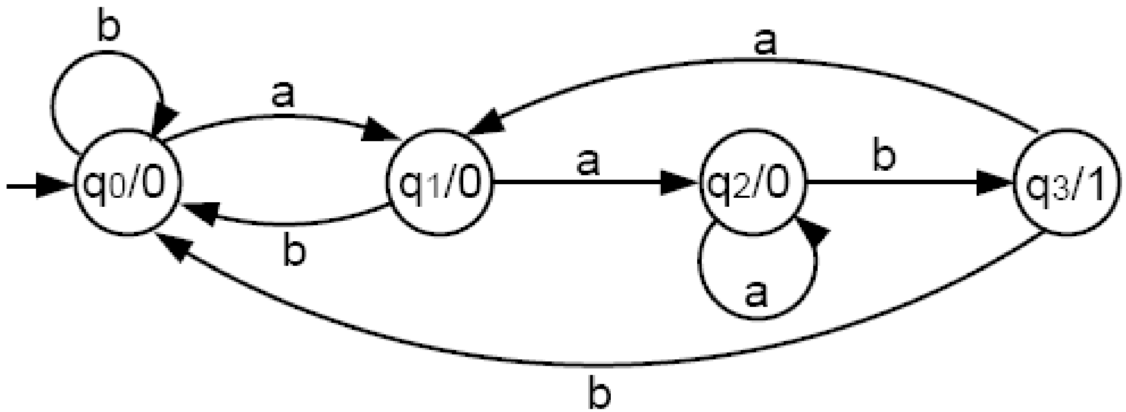
Example



- We indicate the start state by an outside arrow since there is no room for the usual - sign.
- Given the input string *abab*, the output sequence is 10010.
- Note that the length of the output string is one longer than the length of the input string.

Example

- Suppose we are interested in knowing exactly how many times the substring *aab* occurs in a long input string. The following Moore machine will count this for us:



- Every state of this machine prints out a 0, except for q3, which prints a 1.

Example contd.

- To get to q_3, we must have come from q_2 and have just read a b. To get to q_2, we must have read at least two a's in a row.
- After finding the subtring *aab* and tallying a 1 for it, the machine looks for the next *aab*. Hence, the number of 1's in the output string is exactly the number of substrings *aab* in the input string.
- Consider an FA that accepts a language L:
  - If we add printing character 0 to any non-final state and 1 to each final state, then the 1's in any output string mark the end position of all substrings that are words in L.
  - In a similar way, a Moore machine can be said to *define* the language of all input strings whose output ends with a 1.
  - The Moore machine above with q0 = - and q3 = + accepts all words that end with aab.

Melay machine Definition

A **Mealy machine** is a collection of four things:

1. A finite set of states q_0, q_1, q_2, ..., where q_0 is designated as the start state.

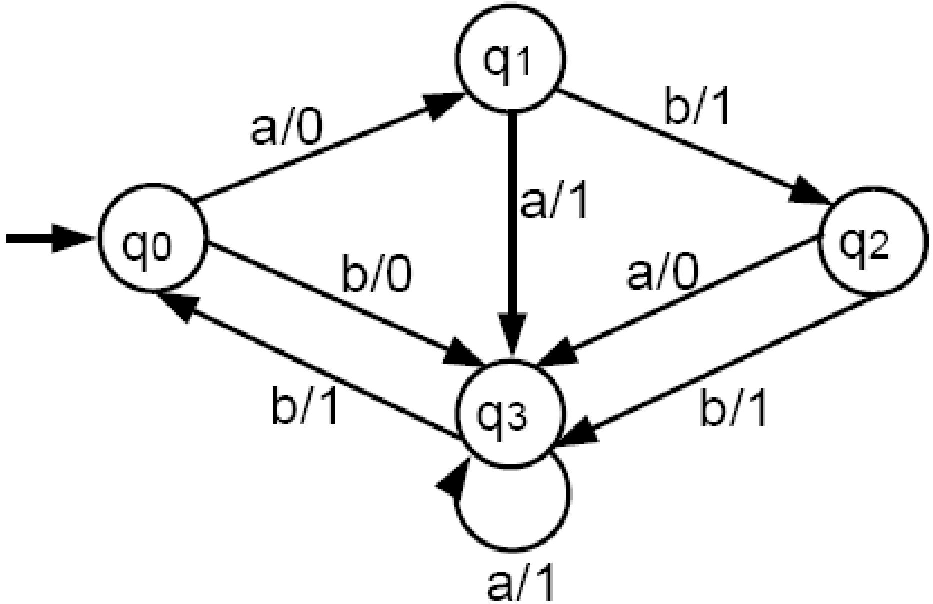
2. An alphabet of letters for forming the input string $\sum = \{a, b, \ldots\}$.

3. An alphabet of possible output characters $\Gamma = \{0, 1, \ldots\}$.

4. A pictorial representation with states represented by small circles and directed edges indicating transition between states.

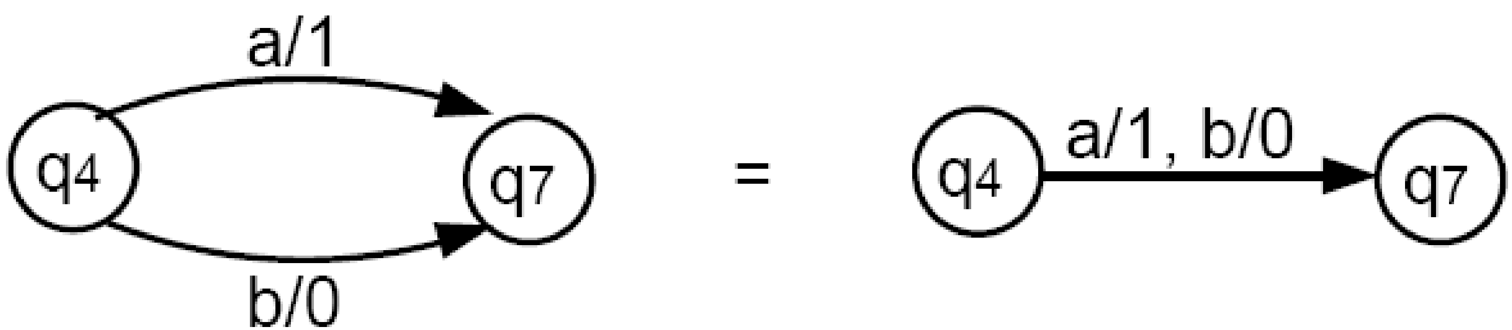
- Each edge is labeled with a compound symbol of the form ***i/o*** where *i* is an input letter and *o* is an output character.
- Every state must have exactly one outgoing edge for each possible input letter.
- The edge we travel is determined by the input letter *i*. While traveling on the edge, we must print the output character *o*.

Example



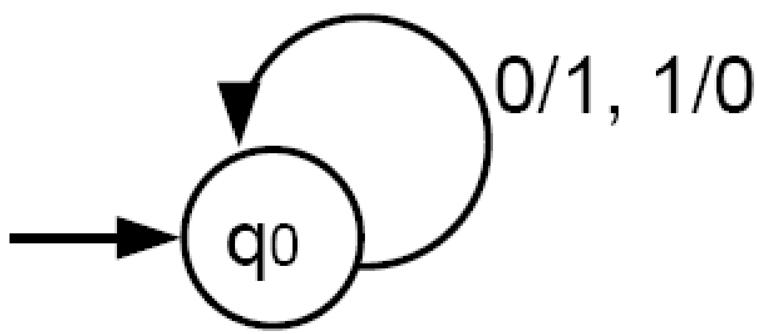
- Given the input string *aaabb*, the output is 01110.
- In a Mealy machine the output string has the same number of characters as the input string has letters.

- A Mealy machine does not define a language by accepting and rejecting input strings: It has no final states.
- However, there is a sense in which a Mealy machine can recognize a language, as we will see later.
- Note the following notation simplification:



Example

- The following Mealy machine prints out the 1's complement of an input bit string.
- This means that it will produce a bit string that has a 1 whenever the input string has a 0, and a 0 whenever the input has a 1.



- If the input string is 001010, the output will be 110101

Example

- Let consider a Mealy machine, called **increment machine**, which reads a binary number and prints out the binary number that is **one larger**.

- Assume that the input bit string is a binary number fed in backward; that is, unit digit first, then 2's digit, 4's digit, etc.

- The output string will be the binary number that is one greater and that is **generated right to left**.

- The machine will have 3 states: start, owe-carry and no-carry. The owe-carry state represents the overflow when two bits of 1's are added, we print a 0 and we carry a 1.

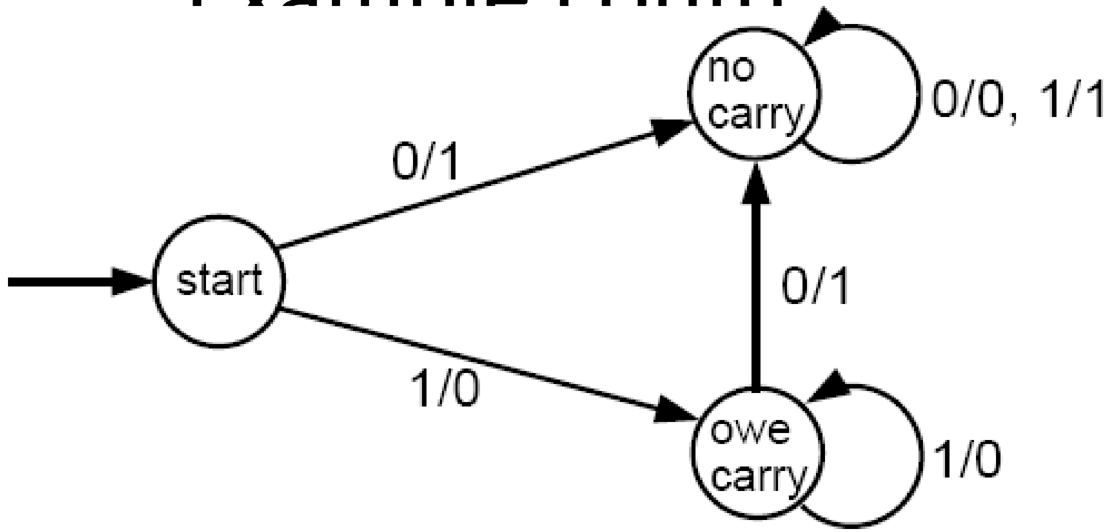
Example contd.

- From the start state, if we read a 0, we print a 1 (incrementing process), and we go to the no-carry state. If we read a 1, we print a 0 (incrementing) and we go to the owe-carry state.

- At any point in the process, in we are in the no-carry state, we print the next bit just as we read it and remains in no-carry.

- However, if we are in the owe-carry state and read a 0, we print a 1 and go to no-carry. If we are in owe-carry and read a 1, we print a 0 and we loop back to owe-carry.

Example contd



- Let the input string be 1011 (binary representation of 11).
- The string is fed into the machine as 1101 (backwards).
- The output will be 0011, which when reversed is 1100 and is the binary representation of 12.
- In Mealy machine, output length = input length. Hence, if input were 1111, then output would be 0000 (**overflow situation**).

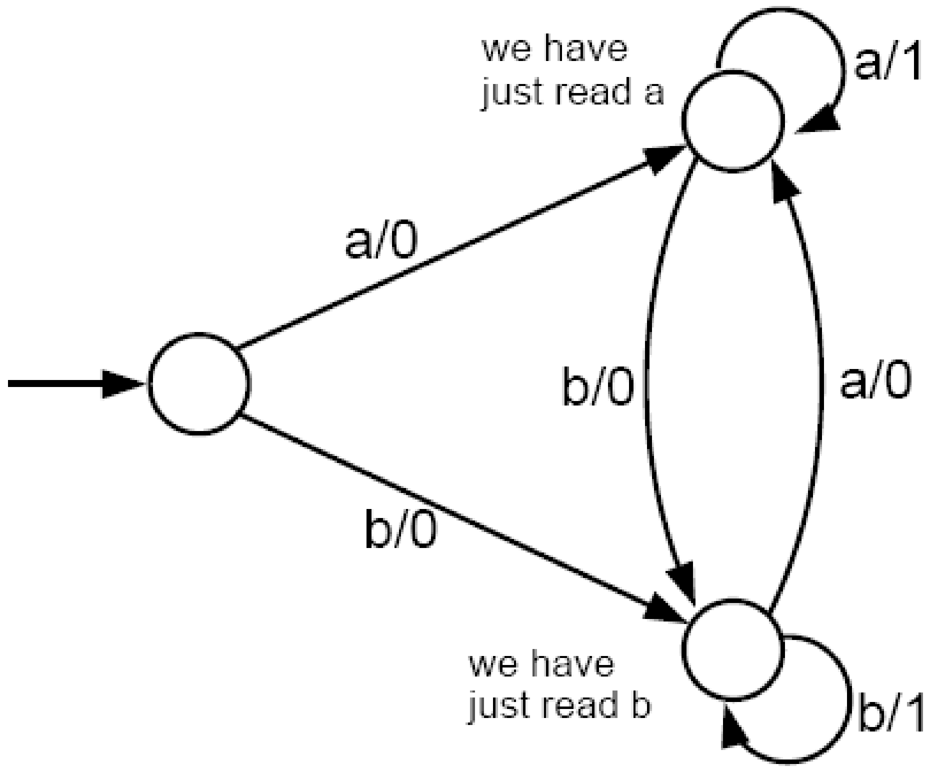
Example

- Although a Mealy machine does not accept or reject an input string, it can recognize a language by making its output string answer some question about the input.

- Consider the language of all words that have a double letter (*aa* or *bb*) in them.

- We can build a Mealy machine that can take an input string of a’s and b’s, and print out an output string of 0’s and 1’s such that **if the n-th output character is a 1, it means that the n-th input letter is the second letter in a pair of double letters**.

- The complete picture of this machine is as follows:



- If the input string is *ababbaab*, the output will be 00001010.
- This machine recognizes the occurrences of *aa* or *bb*.
- Note that the triple letter word *aaa* produces the output 011 since the second and third letters are both the back end of a pair of double a’s.

Moore = Melay

- So far, we have define that two machines are equivalent if they accept the same language.
- In this sense, we cannot compare a Mealy machine and a Moore machine because they are not language definers.

Definition:

- Given the Mealy machine *Me* and the Moore machine *Mo* (which prints the automatic start state character x), we say that these two machines are **equivalent** if for every input string, the output string from *Mo* is exactly x concatenated with the output string from *Me*.

Theorem 8

If Mo **is a Moore machine, then there is a Mealy machine** Me **that is equivalent to** Mo**.**

Proof by constructive algorithm:

- Consider a particular state in *Mo*, say state q_4, which prints a certain character, say *t*.
- Consider all the incoming edges to q_4. Suppose these edges are labeled with a, b, c, ...
- Let us re-label these edges as a/t, b/t, c/t, ... and let us erase the t from inside the state q_4. This means that we shall be printing a t on the incoming edges before we enter q_4.

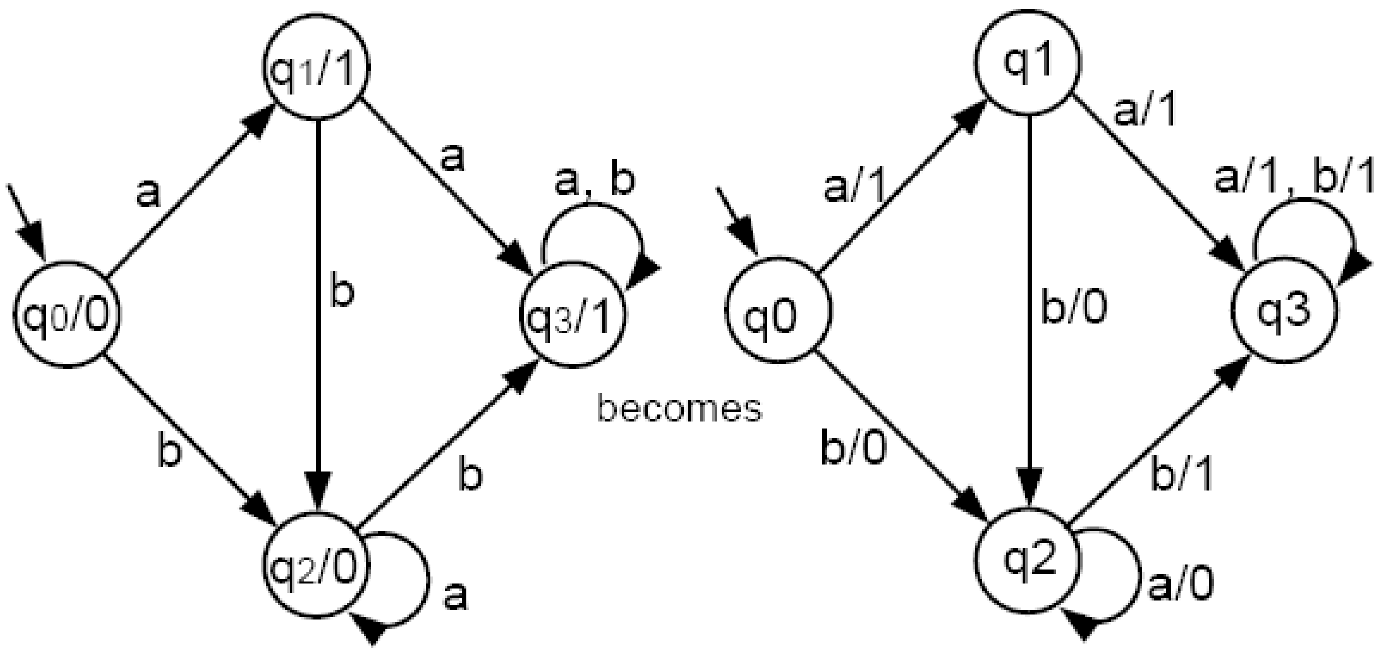
Proof by constructive algorithm contd.

- We leave the outgoing edges from q_4 alone. They will be relabeled to print the character associated with the state to which they lead.

- If we repeat this procedure for every state q_0, q_1, ..., we turn *Mo* into its equivalent *Me.*

Example

- Following the above algorithm, we convert a Moore machine into a Mealy machine as follows:

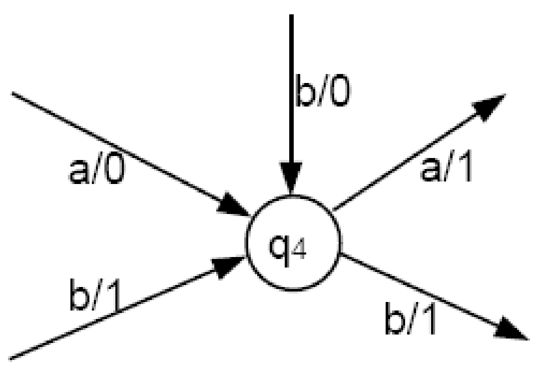


Theorem 9

For every Mealy machine Me**, there is a Moore machine** Mo **that is equivalent to it.**

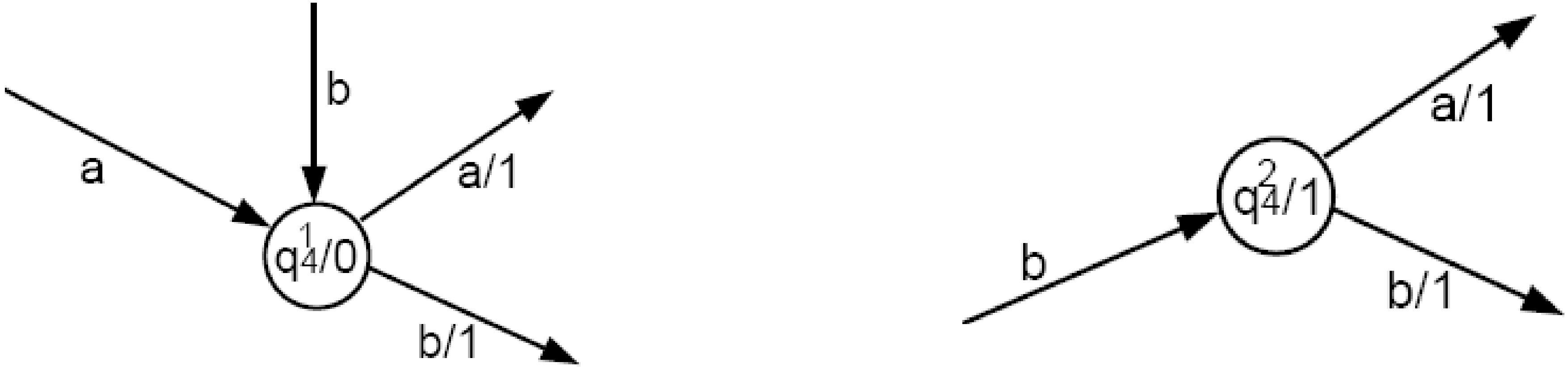
Proof by constructive algorithm:

- We cannot just do the reverse of the previous algorithm. If we try to push the printing instruction from the edge (as it is in Me) to the inside of the state (as it should be for Mo), we may end up with a conflict: Two edges may come into the same state but have different printing instructions, as in this example:



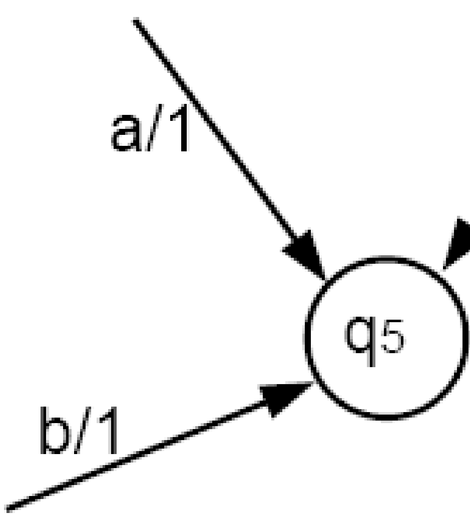
Proof by constructive algorithm (cont.):

- What we need are two copies of q_4, one that prints a 0 (labeled as $q^1{}_4/0$), and the other that prints a 1 (labeled as $q^2{}_4/1$). Hence,
  - The edges a/0 and b/0 will go into $q^1{}_4/0$.
  - The edge b/1 will go into $q^2{}_4/1$.
- The arrow coming out of each of these two copies must be the same as the edges coming out of q_4 originally.

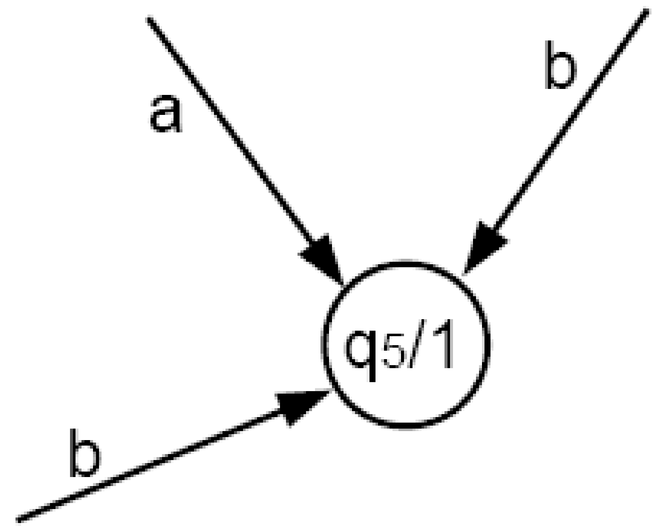


Proof by constructive algorithm (cont.):

- If all the edges coming into a state have the same printing instruction, we simply push that printing instruction into the state.

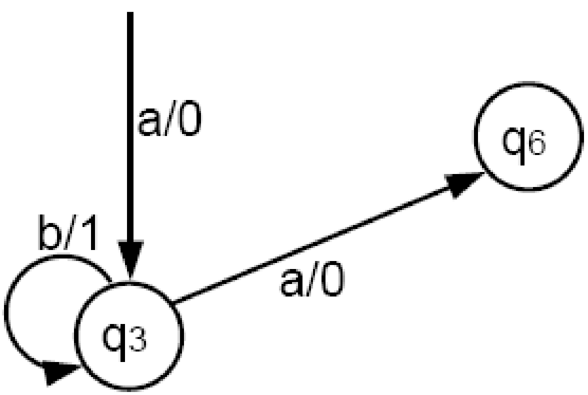


becomes

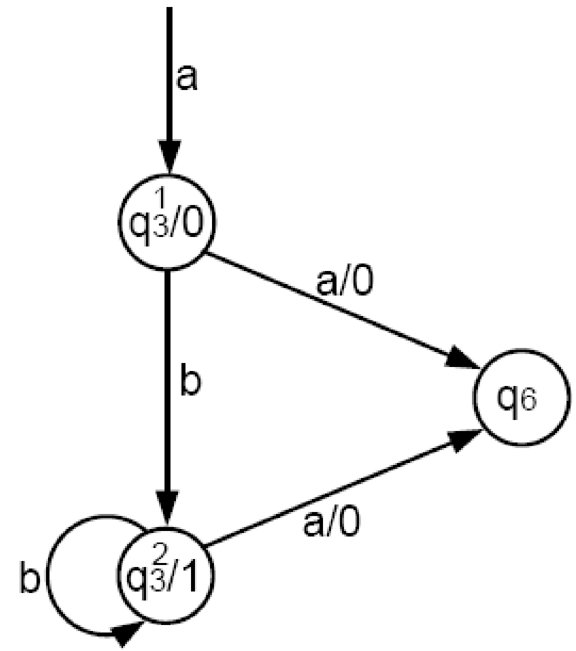


Proof by constructive algorithm (cont.):

- An edge that was a loop in *Me* may becomes two edges in *Mo*, one that is a loop and one that is not.



becomes

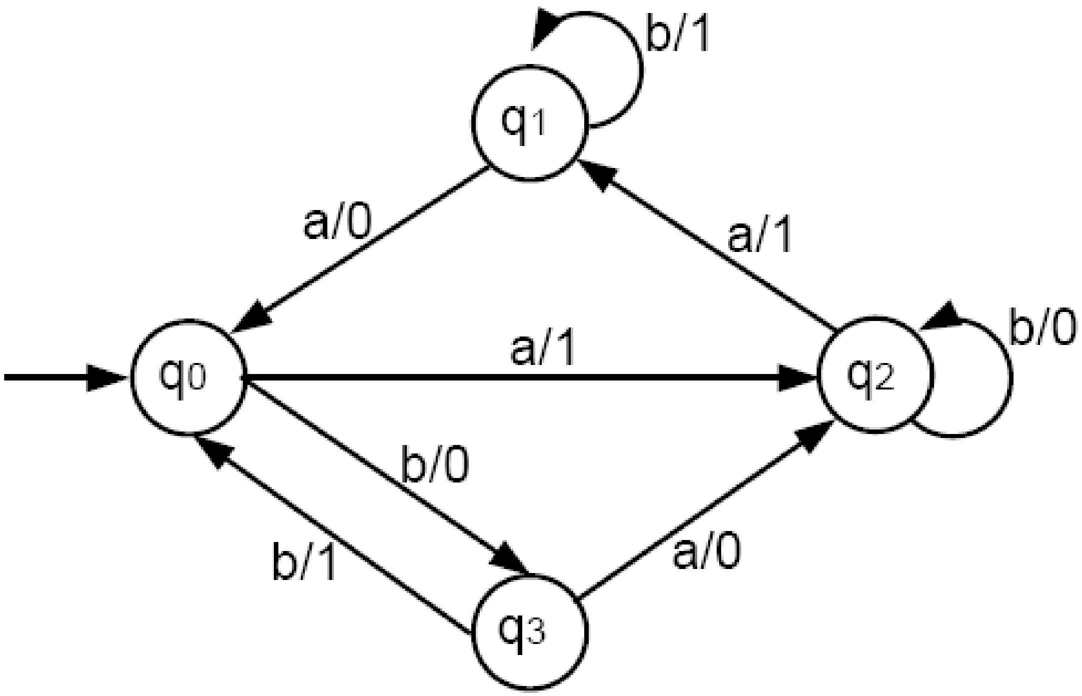


Proof by constructive algorithm (cont.):

- If there is ever a state that has no incoming edges, we can assign it any printing instruction we want, even if this state is the start state.
- *If we have to make copies of the start state in Me, we can let any of the copies be the start state in Mo, because they all give the identical directions for proceeding to other states.*
- Having a choice of start states means that the conversion of *Me* into *Mo* is NOT unique.
- Repeating this process for each state of Me will produce an equivalent Mo. The proof is completed.
- **Together, Theorems 8 and 9 allow us to say** Me = Mo**.**

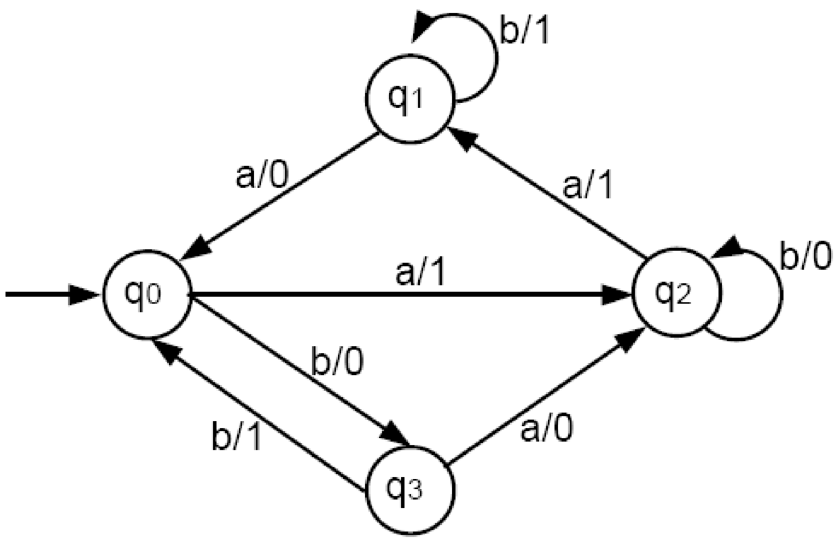
Example

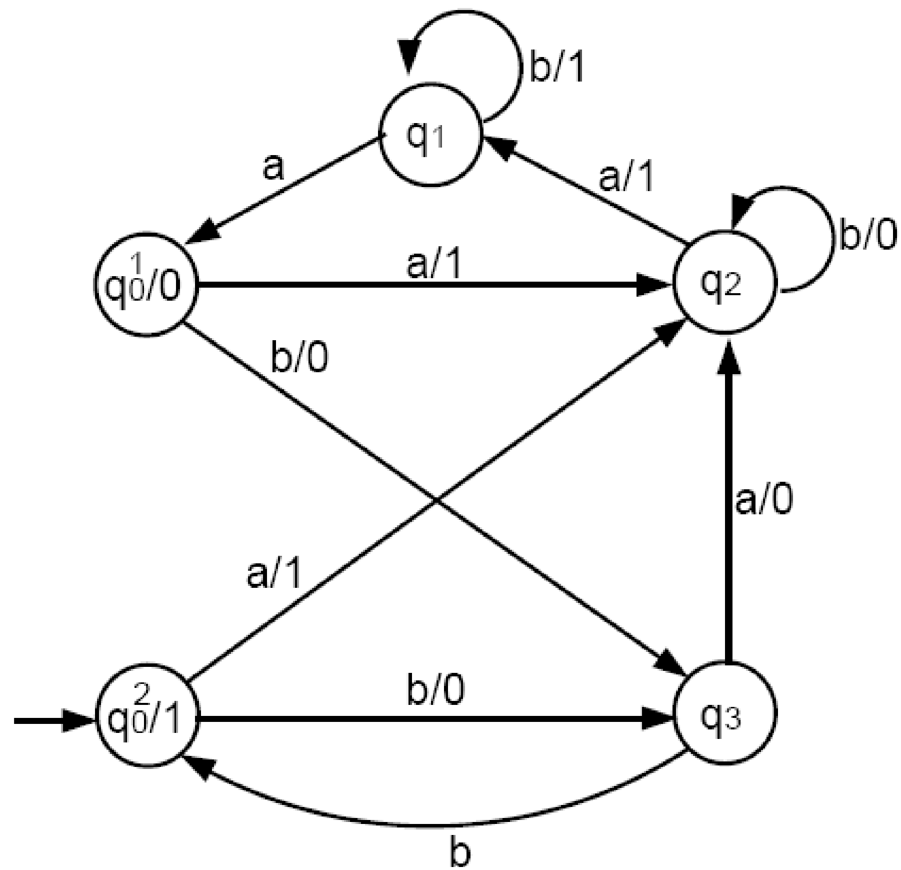
- Convert the following Mealy machine into a Moore machine:



Example contd.

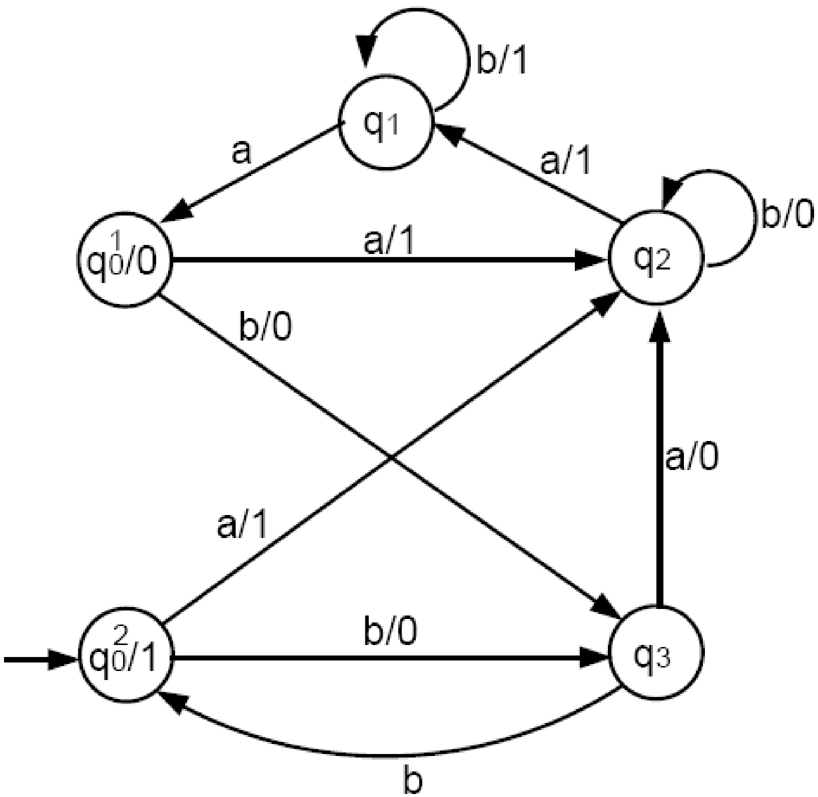
- Following the algorithm, we first need two copies of q_0:

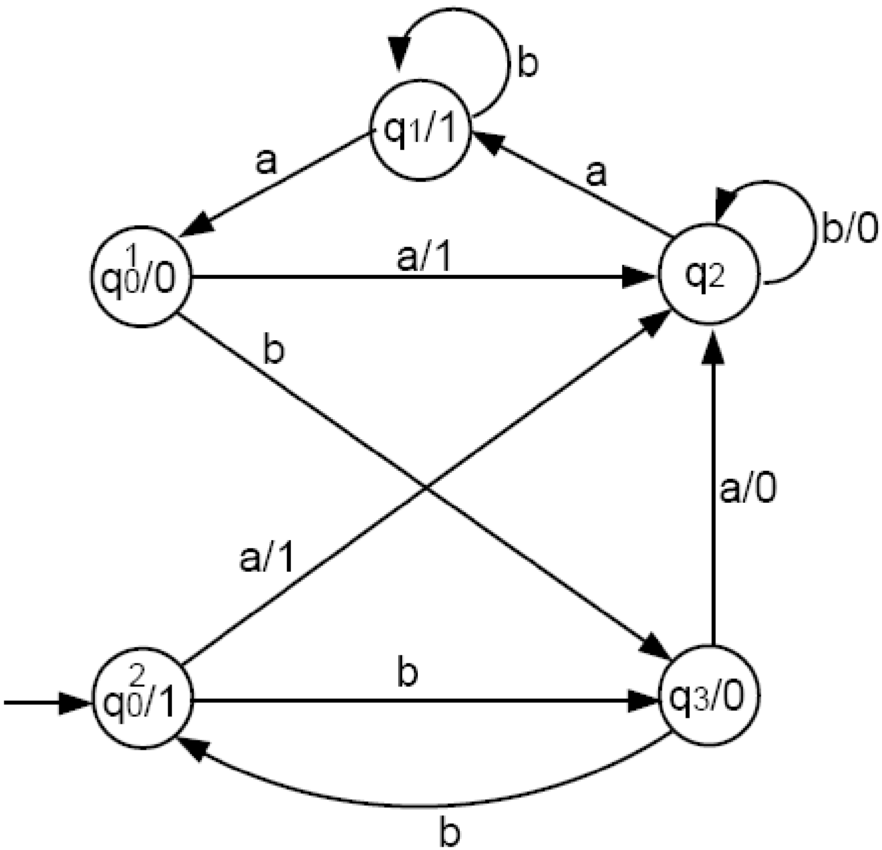




Example contd.

- All the edges coming into state q_1 (and also q_3) have the same printing instruction. So, apply the algorithm to q_2 and q_3:





Example contd.

- The only job left is to convert state q_2. There are 0-printing edges and 1-printing edges coming into q_2. So, we need two copies of q_2. The final Moore machine is: