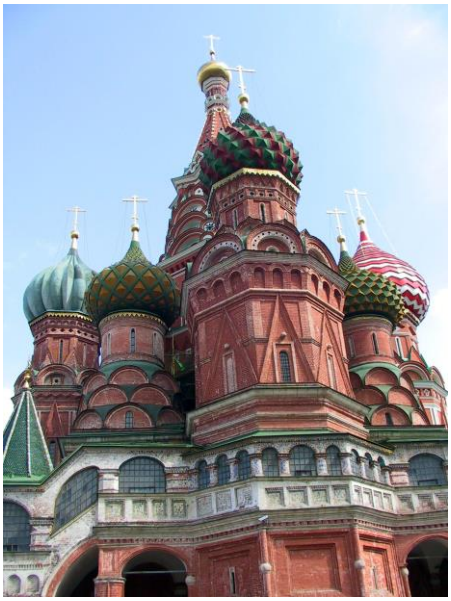
Chapter 4S

Sequential Machines

(2015/10/22)



Moscow, Russia

Outline

4s.1 Basic Concepts of Sequential Machine

- **Concepts** ---
  - Each FA accepts inputs but yields no output, or we can say, yields the simple outputs of "accept" or "reject."
  - Many application systems have both inputs and outputs, like digital systems, computers, compilers, etc.
  - We need automata with capabilities of not only accepting inputs but also yielding outputs --- *sequential machines*!

- **Basic properties of sequential machines** ---
  - The sequential machine may be regarded as an extension of the deterministic finite automaton.
  - The name "sequential machine" comes from its nature of changing its output and state "in a timely order," or equivalently, "sequentially," according to the input and the current state.
  - The sequential machine is defined to have *no final state*.
  - The sequential machine has many applications for systems with outputs, like modeling vending machines, elevators, traffic signals, etc.

- Abbreviation of a term --- in the sequel, we abbreviate "sequential machine" as "SM."

4s.2 Definition of Sequential Machine

- **Types of sequential machine** ---

  There are two types of SM --- *Mealy machine* and *Moore machine*, which were developed from the study of digital systems.

  - Fig. 4s.1 shows the Mealy machine model whose output depends on *both the input and the current state* of the machine.
  - Fig. 4s.2 shows the Moore machine model whose output depends *only on the current state* of the machine.

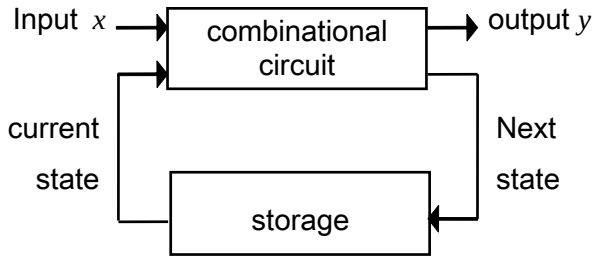


Fig. 4s.1 Mealy machine model.

- **Example 4s.1 (a serial adder – a Mealy machine)** ---

  Fig. 4s.3 is a *serial adder* commonly seen in digital systems, which adds sequentially three 1-bit inputs (the addend x, augend y, and the old carry c) and yields two 1-bit outputs (the sum z and the new carry c').

- For example, in the series addition result 0011 + 1001 = 1100 with carries 0011, the serial adder adds up initially $x = 0$, $y = 1$, $c = 0$ (the initial value if the flipflop) to get $z = 1$, $c' = 0$.
- The serial adder obviously can be checked to be a Mealy machine because comparing with Fig. 4s.1, we have
  - the *full adder* = the combinatorial circuit in Fig. 4s.1;
  - the D-type flipflop = the storage in Fig. 4s.1;
  - the addend x, the augend y, and the old carry c = the input;
  - the sum z and the new carry c' = the output;
  - the new carry c' and the pulse signal s (for triggering the flipflop) = the next state;
  - the old carry c = the current state.

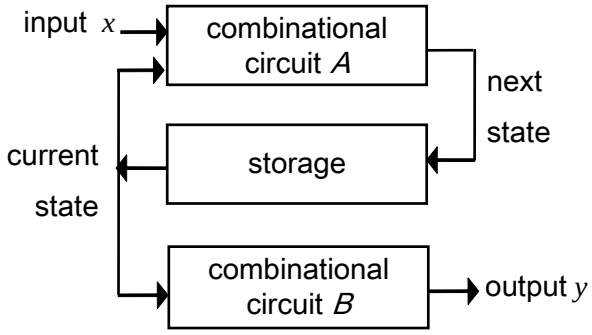


Fig. 4s.2 Moore machine model.

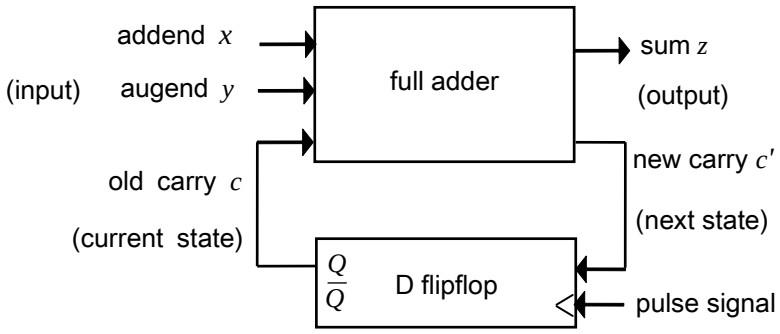


Fig. 4s.3 Full adder --- a Mealy machine --- of Example 4s.1.

■ **Example 4s.2 (a 2-bit binary counter with outputs - a Moore machine) ---**

Fig. 4s.4 is a 2-bit *binary counter*, which counts binary numbers 00, 01, 10, 11 in sequence and cyclically in accordance with the input pulse signals.

- The counter has a 1-bit output of 1 when the count comes to the number 00, so that it has an output sequence of the form 00010001… if the input pulse signal continues for a long time.
- The binary counter can be checked to be a Moore machine because comparing with Fig. 4s.2, we have
  - the two D-type flipflops = the storage in Fig. 4s.2;
  - the circuit to the right of the two flipflops = combinational circuit A in Fig. 4s.2;

- the circuit to the left of the two flipflops = combinatorial circuit B in Fig. 4s.2;
- the output *y* only depends *only* on the state of the two flipflops.

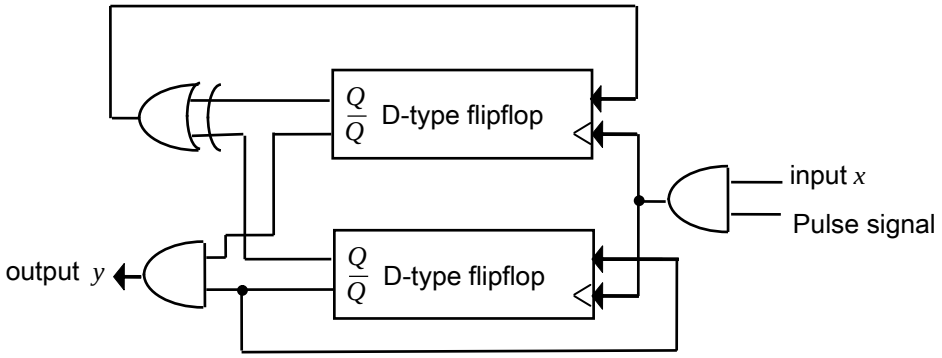


Fig. 4s.4 Binary counter --- a Moore machine --- of Example 4s.2.

■ **Definition 4s.1 (Mealy machine)** ---

A Mealy machine M is a 6-tuple $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$, where

1. Q, Σ, δ and q_0 are the same as defined for the DFA;
2. Δ is a finite, non-empty output alphabet;
3. λ is an output function such that λ: $Q \times \Sigma \rightarrow \Delta$, i.e., for any $q \in Q$ and any $a \in \Sigma$, $\lambda(q, a) = b$ where $b \in \Delta$.

■ **Definition 4s.2 (Moore machine)** ---

A Moore machine M is a 6-tuple $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$, where all elements are the same as those defined for the Mealy machine except that λ: $Q \rightarrow \Delta$ (instead of λ: $Q \times \Sigma \rightarrow \Delta$), i.e., for any $q \in Q$, $\lambda(q) = b$ where $b \in \Delta$.

■ **Discussions on the definitions** ---

♦ Determinism of SM's ---

The types of SM's are both deterministic in nature as defined.

♦ Difference between the two types of SM's ---

The output of the Mealy machine depends on the current state and the input while that of the Moore machine depends only on the current state.

- But this difference does not cause any difference on the capabilities of the two types and this will be proven later in this chapter.

♦ The output of the Mealy machine ---

Given an input string $x = a_1a_2\ldots a_n$, the output of the Mealy machine is another string of the form $\lambda(q_0, a_1)\lambda(q_1, a_2)\ldots\lambda(q_{n-1}, a_n)$, where $q_1, q_2, \ldots, q_{n-1}$ result from a series of state transitions $\delta(q_{i-1}, a_i) = q_i$ starting from the initial state q_0, where $i = 1$,

2, ….

- We use $\lambda(x)$ to represent the output string $\lambda(q_0, a_1)\lambda(q_1, a_2)\ldots\lambda(q_{n-1}, a_n)$.

♦ The output of the Moore machine ---

Given the input string $x = a_1a_2\ldots a_n$, the output of the Moore machine is another string $\lambda(q_0)\lambda(q_1)\ldots\lambda(q_{n-1})$, where $q_1, q_2, \ldots, q_{n-1}$ are generated in the same way as the Mealy machine does.

- Similarly, we use $\lambda(x)$ to represent the output string $\lambda(q_0)\lambda(q_1)\ldots\lambda(q_{n-1})$.

♦ Response of the Moore machine to the initial state ---

Because the Moore machine yields an output in accordance with the current state only, it has an output in the initial state, which is just $\lambda(q_0) \in \Delta$.

- This may be regarded as an output $\lambda(q_0)$ for the input of the empty string ε.

♦ Response of the Mealy machine to the initial state ---

As a contrast to the former case, the Mealy machine does not yield an output in its initial state because only when there is an input can the Mealy machine has an output.

- Therefore, we may say that for the input of the empty string ε, the Mealy machine yields the output of the empty string ε without changing the state. This corresponds to the case that the extended DFA has the transition $\delta(q_0, \varepsilon) = q_0$.

♦ Final state of the SM ---

The SM yields an output symbol each time an input symbol is supplied; *no* final state is involved in this process.

♦ A graphic model for the SM ---

A graphic model for the SM is shown in Fig. 4s.5, which is the same as that for the DFA except that there is an additional tape on which the output string of symbols are written with a writing head.

♦ Descriptions of the SM ---

Like the DFA, the *state transition function* and the *output function* of an SM can be described by an exhaustive description, a state transition diagram, or a state transition table which includes additionally mappings to the outputs, as shown by the following two examples.

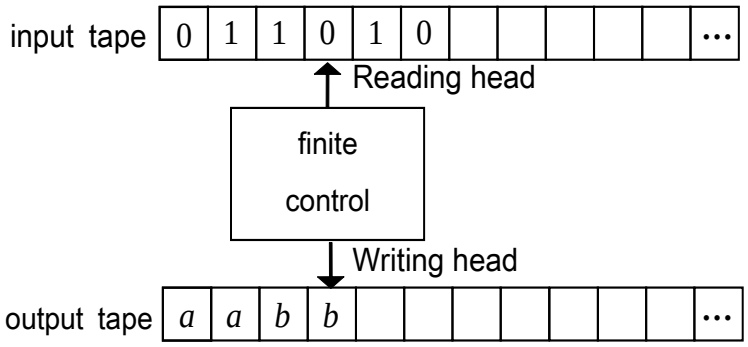


Fig. 4s.5 A graphic model for the sequential machine.

■ **Example 4s.3 (descriptions of a Mealy machine – the full adder)** ---

The various descriptions of the full adder, which is a Mealy machine M as mentioned in Example 4s.1, are given in the following:

$$M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$$

where

- $Q = \{0, 1\}$(the 1-bit carry c kept in the D-type flipflop represents the state of the machine);
- $\Sigma = \{00, 01, 10, 11\}$(the bit pair consisting of the values x and y of the addend and augend, respectively, is the input);
- $\Delta = \{0, 1\}$(the sum z is the output);
- δ: $Q \times \Sigma \rightarrow Q$, or more specifically, δ: $(c, xy) \rightarrow c'$ where c' is the carry computed by use of the full adder;
- λ: $Q \times \Sigma \rightarrow \Delta$, or more specifically, λ: $(c, xy) \rightarrow z$;
- q_0 is the initial binary value kept in the D-type flipflop, which is assumed to be 0;
- δ and λ are described by three ways as follows.
  - Exhaustive listing ---

    $\delta(0, 00) = 0$; $\delta(0, 01) = 0$; $\delta(0, 10) = 0$; $\delta(0, 11) = 1$;
    $\delta(1, 00) = 0$; $\delta(1, 01) = 1$; $\delta(1, 10) = 1$; $\delta(1, 11) = 1$;
    $\lambda(0, 00) = 0$; $\lambda(0, 01) = 1$; $\lambda(0, 10) = 1$; $\lambda(0, 11) = 0$;
    $\lambda(1, 00) = 1$; $\lambda(1, 01) = 0$; $\lambda(1, 10) = 0$; $\lambda(1, 11) = 1$.

    e.g., $\delta(1, 01) = 1$ represents the binary addition $c + (x + y) = 1 + (0 + 1) = 10$ which means a sum $z = 0$ and a new carry $c' = 0$.

  - Transition diagram --- as shown in Fig. 4s.6 where the notation xy/z beside each arrow means that with the input xy, the machine yields the output z.

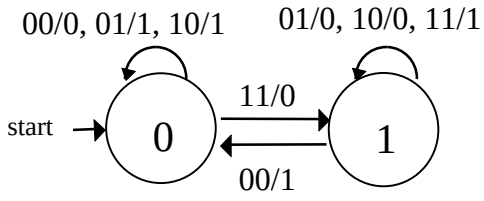


Fig. 4s.6 Transition diagram of the Mealy machine --- a full adder --- of Example 4s.3.

  - Transition table --- as shown in Table 4s.1 where the notation c'/z in each entry means that the machine gets into the new state c' and yields the output z.

■ **Example 4s.4 (descriptions of a Moore machine – the binary counter)** ---

The various descriptions of the binary counter of Example 4s.2, which is a Moore machine M, are given in the following:

Table 4s.1 Transition table of Example 4s.3.

state \ input	00	01	10	11
0	0/0	0/1	0/1	1/0
1	0/1	1/0	1/0	1/1

$$M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$$

where

- $Q = \{00, 01, 10, 11\}$(the bit pair uv kept in the two flipflops represents the state of the machine);
- $\Sigma = \{0, 1\}$(input $x = 0$ means stopping counting; input $x = 1$ means beginning counting);
- $\Delta = \{0, 1\}$(output $y = 0$ means the count value of 1, 2, or 3; output $y = 1$ means the count value of 0);
- δ: $Q \times \Sigma \rightarrow Q$, or more specifically, δ: $(uv, x) \rightarrow u'v'$ where bit pair $u'v'$ represent the new state;
- λ: $Q \rightarrow \Delta$, or more specifically, λ: $uv \rightarrow y$;
- q_0 specifies the initial bit pair kept in the two flipflops, assumed to be 00;
- δ and λ are described by two ways as follows (the exhaustive listing is omitted).
  - Transition diagram --- as shown in Fig. 4s.7 where the notation uv/y inside each state circle means that the machine in state uv yields the output y.
  - Transition table --- as shown in Table 4s.2 where the notation uv/y in each entry of the first column means the same as that used in the transition diagram.

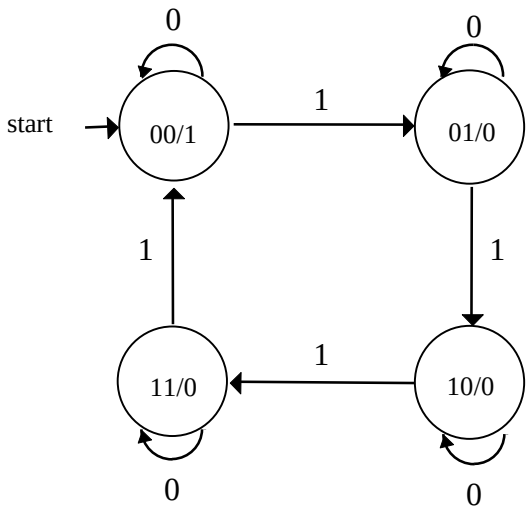


Fig. 4s.7 Transition diagram of the Moore machine --- a binary counter --- of Example 4s.4.

Table 4s.2 Transition table of Example 4s.4.

Current state/output \ input	0	1
00/1	00	01
01/0	01	10
10/0	10	11
11/0	11	00

■ **State transition tables used in digital systems ---**

For digital systems, automata are described by transition tables of another form which has the merit that there is no difference between the table for the Mealy machine and that for the Moore machine. We call such transition tables to be *of the digital system style*.

♦ The digital-system-style transition table for the fuller adder of Example 4s.3 is shown in Table 4s.3 and that for the binary counter of Example 4s.4 is shown in Table 4s.4.

Table 4s.3 The transition table of the digital system style of Example 4s.3.

Current state	Input *x*	Next state	Output *y*
0	00	0	0
0	01	0	1
0	10	0	1
0	11	1	0
1	00	0	1
1	01	1	0
1	10	1	0
1	11	1	1

Table 4s.4 The transition table of the digital system style of Example 4s.4.

Current state	Input *x*	Next state	Output *y*
00	0	00	1
00	1	01	1
01	0	01	0
01	1	10	0
10	0	10	0
10	1	11	0
11	0	11	0
11	1	00	0

■ **Example 4s.5** ---

Redesign of the vending machine described in Chapter 0 with a Mealy machine.

- ♦ In Chapter 0, we have used a DFA to design a drink-selling vending machine with no output which is not realistic because a real vending machine does have outputs, the sold drinks.
- ♦ Here the machine will be remodeled by an SM, or more specifically, by a Mealy machine.
- ♦ The DFA in Chapter 0 is redrawn here as Fig. 4s.8with:
  - all the dollar signs in the diagram being removed,
  - the notation $*i* of each state being changed into <*i*> for clarity, and
  - the two transitions from state 15 to state 20 being combined into one (represented with only one arrow now).
- ♦ The machine only sells 20-dollar goods with 5- and 10-dollar coins as inputs; and it has several types of outputs:
  - coins *other than 5 and 10 dollars* which are not accepted by the machine, as assumed, to simplify the design;
  - the *excessive amount of money* which will be returned when more than 20 dollars are put into the machine slot;
  - the *drink dropped out* which the user choose to buy by pushing a "buy button";
  - *no response* from the machine which the input money is not sufficient to buy a drink.

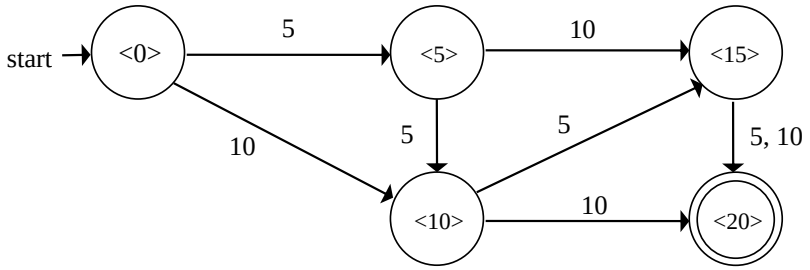


Fig. 4s.8 The vending machine modeled by an DFA (a repetition of Fig. 1.2).

- ♦ The desired Mealy machine is described as follows:

$$M = (Q, \Sigma, \Delta, \delta, \lambda, q_0) \text{ ,}$$

where

1. $Q = \{0, 5, 10, 15, 20\}$ in units of dollar;
2. input alphabet $\Sigma = \{5, 10, b\}$ with
   - "5" and "10" representing respectively 5- and 10-dollar coins, and
   - "*b*" representing pushing the "buy button";
3. output alphabet $\Delta = \{\varphi, 5, 10, f\}$ with
   - φ representing "no response" from the vending machine;
   - "5" and "10" representing the returned coin changes; and
   - *f* representing the drink dropped out of the machine;

4. δ and λ are shown in Fig. 4s.9.
5. the initial state $q_0 = 0$.

♦ The differences between Fig. 4s.8 and Fig. 4s.9 include the following.
1. No final state is used now in the Mealy machine.
2. Pushing the "buy button" (i.e., with the input *b*) while the machine is in all states other than <20> will cause the machine to emit an output of "no response" without changing the state.
3. Putting money into the machine while it is in states other than <15> and <20>, will cause the machine to change its state but with no output;
4. Putting 5 dollars into the machine will cause no output, and putting 10 dollars into to the it will result in a return of 5 dollars; and for both cases the machine will get into the state of <20>."
5. In the state of <20>, all money put into the machine will be returned, and in that state, a push of the buy button" will cause a drink to drop out and a return into the initial state.

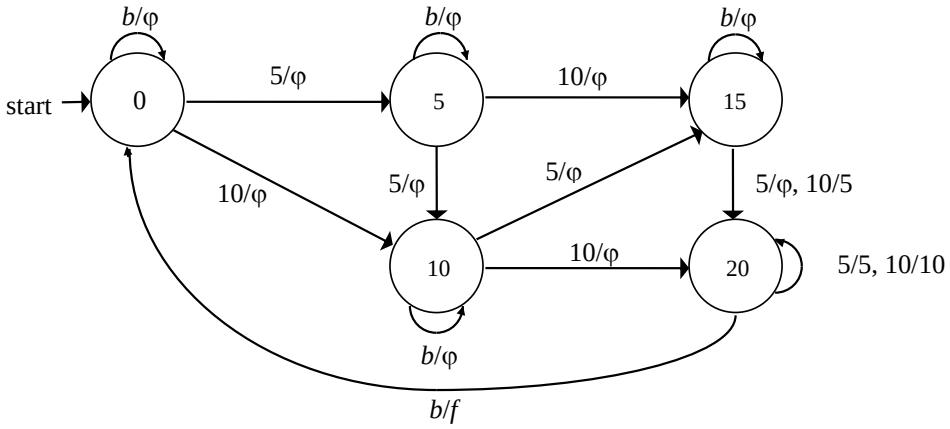


Fig. 4s.9 The vending machine modeled by a Mealy machine.

4s.3 Implementation of Finite Automata by Sequential Machines

■ **Concept** ---
♦ Though a DFA has no output, it has the simple responses of "accept" and "reject" to input strings.
♦ If we regard these two types of responses as outputs, and use the output strings of "1" and "0" to represent them respectively, then we can use the SM to implement a DFA.

■ **Implementation of a DFA by an SM** ---
♦ Let $M = (Q, \Sigma, \delta, q_0, F)$ be a given DFA, we can implement it by two ways using the Mealy machine and the Moore machine.

♦ *Implementation by a Moore machine* ---

$$M' = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$$

where

- $\Delta = \{0, 1\}$(with "1" representing "accept," and "0" representing "reject");
- λ is defined as: for all $q \in F$, $\lambda(q) = 1$, and for all $q \notin F$, $\lambda(q) = 0$ (i.e., if the machine gets into a final state, it emits "1" as the output; otherwise, "0" as the output).

♦ Pros and cons of using the Moore machine ---

- Merit --- the machine, *after getting* into a final state (*not during* state transiting), emits a "1" which emulates the function of the FA to accept the input string.
- Disadvantage 1 --- the machine, whenever in a final state but *with the input string not read to the end*, will emit a "1" to accept the string as well, which obviously is *incorrect* as viewed from the definition of FA.
- Disadvantage 2 --- furthermore, in the initial state q_0, even when the first symbol of the input string is *not* read yet, the machine has the first output of $\lambda(q_0)$ which is *incorrect, either,* as viewed from the definition of FA again.
- We will remove the above two disadvantages later.

♦ *Implementation by a Mealy machine:*

$$M' = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$$

where

- $\Delta = \{0, 1\}$(with "1" representing "accept," and "0" representing "reject");
- λ is defined as: for all $p \in Q$, and for all $a \in \Sigma$, if $\delta(p, a) = q \in F$, then $\lambda(p, a) = 1$; otherwise, $\lambda(p, a) = 0$ (i.e., when a final state is reached from a state, the output "1" is emitted during the state transition; otherwise, "0" is emitted).

♦ Pros and cons of using the Mealy machine ---

- Merit 1 --- the machine, *while transiting* into a final state, emits an "accept" signal "1" as output, unlike the Moore machine above which emits the "accept" signal after getting into the final state.
- Merit 2 --- no output is emitted while in the initial state because the Mealy machine emits output symbols during state transitions.
- Disadvantage --- similar to Disadvantage 1 of the Moore machine above, i.e., the machine, whenever transiting into a final state, will emit a "1" to accept the string even if the input string is *not read to the end* (an *incorrect* action again as viewed from the definition of FA).

■ **Example 4s.6** ---

Implement the DFA given in Fig. 4s.10(a) by a Mealy machine and a Moore machine. (Can you see what language the DFA accepts?)

♦ *Implementation by a Mealy machine* ---

- By the above discussions, we can transform the DFA into a Mealy machine as shown in Fig. 4s.10(b).
- But this Mealy machine has the above-mentioned disadvantage of entering the final state with the emission of an "accept signal" of "1" *before the input string is read to its end.*
- For example, with the input $x = baba$, the output string will be $y = 0101$, in which the first "1" is not what we want.
- One way to remove this advantage is to append an additional special symbol <EOS>

(EOS means the end of a string), and redefine the output function λ to be: only when the state of q_2 is entered *and* the symbol <EOS> is read in can the output "1" be emitted.

- Accordingly, the new Mealy machine becomes that shown in Fig. 4s.10(c).
- And then, with the new input string of x = *baba*<EOS>, the correct output y = 00001 will be emitted as can be checked.

♦ *Implementation by a Moore machine* ---

- If implementation of the DFA by a Moore machine is desired and the above-mentioned disadvantage of entering the final state with the emission of an "accept signal" of "1" *before the input string is read to its end* is to be avoided, then, as can be figured out, it is needed to add an extra state q_3 and redefine the state transition function δ to have the following transition:

  while in the state of q_2, with input symbol <EOS>, the machine enters the state q_3 and emits an "accept" signal as output in that state.

- A Moore machine with the above function is shown in Fig. 4s.10(d).
- With the input string x = *baba*<EOS>, the output string will be y = 000001, in which except the first "0" which is emitted while the machine is in the initial state, the remaining part is all what we want.
- It is noted that the above Moore machine in Fig. 4s.10(d) can be obtained directly from transitioning Fig. 4s.10(c) according to a theorem to be given later.

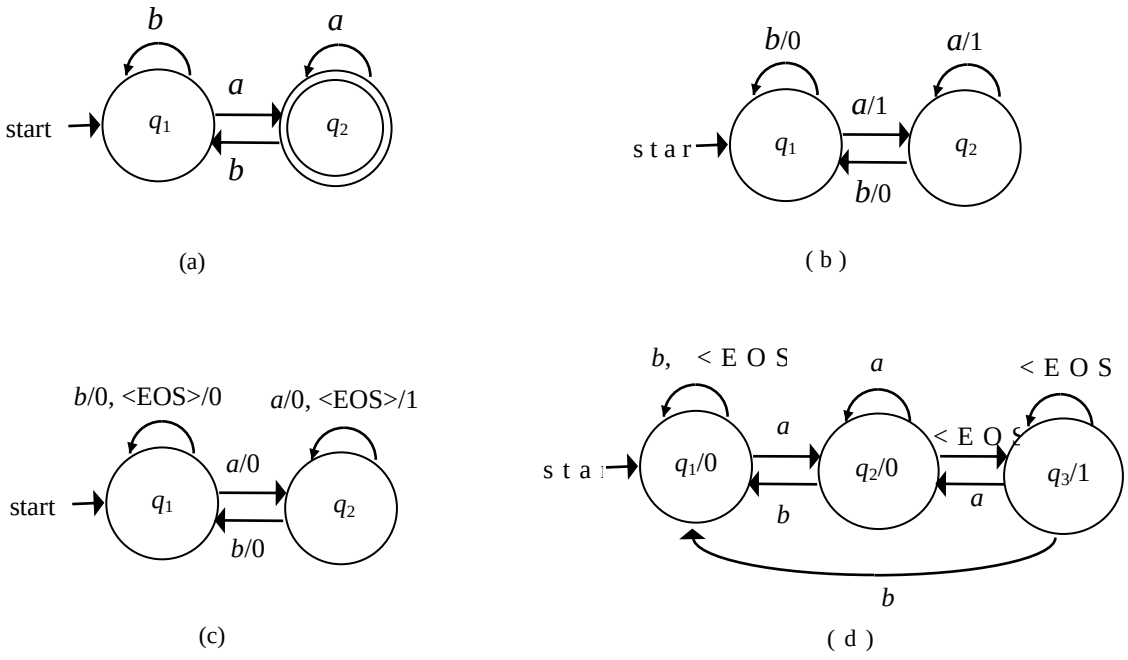


Fig. 4s.10 Transforming a DFA into an SM. (a) A given DFA. (b) A Mealy machine corresponding to (a). (c) A Mealy machine resulting from modifying (b). (d) A Moore machine corresponding to (a).

■ A comparison of the Mealy machine and the Moore machine ---

♦ In the last example, we have seen the fact that with the same functions, the Moore machine will use one more state than the Mealy machine.

- In more general cases, this fact is also true, but not just using one additional state but possibly more.
- In short, compared with the Mealy machine, the Moore machine has a simpler output function but uses more states to solve the same problem.

4s.4 Construction of Deterministic Finite automata by Digital Circuits

■ **Concepts ---**

- The SM was originally developed in the study of digital systems, as mentioned before.
- Therefore, as long as a DFA is implemented by an SM, we can construct the SM with digital circuits to realize the goal of hardware implementation.
- This is really the greatest application of the study of finite automata.

■ **Process of Constructing a DFA by a digital circuit** ---

- In the general case, during the process of solving a digital circuit problem, the first concept of the circuit system to be designed is usually based on an intuitive RE (regular expression) or the ε-NFA (epsilon-nondeterministic finite automaton).
- That is, usually the design process is based on the following steps (the arrows indicate transformations or implementations):

$$\text{concept} \Rightarrow \text{RE} \Rightarrow \varepsilon\text{-NFA} \Rightarrow \not\varepsilon\text{-NFA} \Rightarrow \text{DFA} \Rightarrow \text{SM} \Rightarrow \text{hardware}.$$

- The last step in the above process is the topic of this section.

■ **Example 4s.7** ---

Design a digital circuit to implement the Moore machine of the last example shown in Fig. 4s.10(d).

- A note: when an input string is read by an SM, it is always assumed that the machine begins to work from an initial state.

- *Modification before hardware implementation* ---
  - When a digital system is started, the flipflops in the system will get into arbitrary states, i.e., the values of the flipflops will be random.
  - To remedy this, it is desired to bring the machine from such an arbitrary state into a *specified real initial state*.
  - One way to do this is to add a special symbol <SOS> to the input string as its prefix (SOS means start of string), and then redefine the state transition function to include a new transition: *the machine, in any state, will get into the real initial state* q_1 as long as the special symbol <SOS> is read as an input symbol.
  - Furthermore, so far three states are created for the Mealy machine which we desire. Therefore, two flipflops are needed to keep the three states.
  - But then four states can be created by the use of the two flipflops, yielding a fourth state into which the system might also get when the system is initiated.
  - Therefore, the transition function must also take care of this case, i.e., the transition $\delta(q_4, \text{<SOS>}) = q_1$ must be defined as well.

♦ According to the above discussions, we now can transform Fig. 4s.10(d) into a transition diagram as shown in Fig. 4s.11.

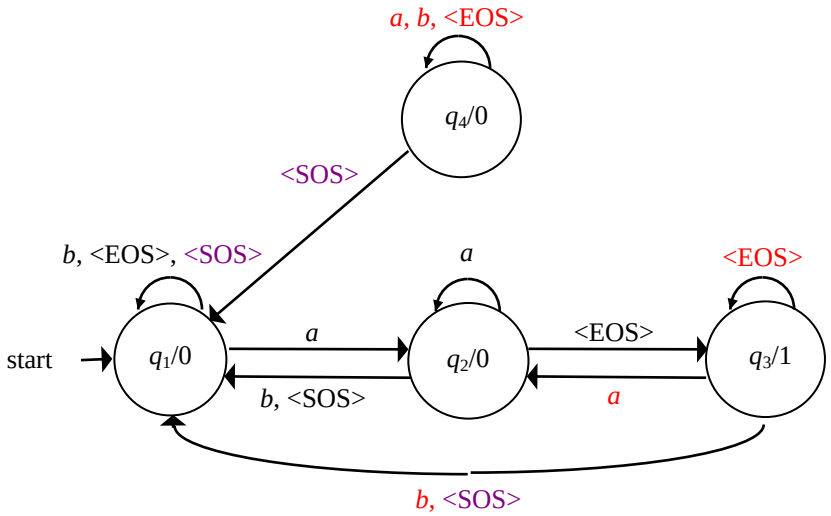


Fig. 4s.11 A new Moore machine resulting from modifying Fig. 4s.10 before being implemented by a digital circuit. (Red symbols are don't cares.)

♦ *Further modification to simplify hardware circuit complexity* ---

- Fig. 4s.11 may be modified further to simplify the hardware circuit we are going to design.
- Specifically, at first we know that after reading an acceptable input string with <EOS> as the last symbol, the machine will get into the state q_3 and emit the accept signal "1."
- Here, we may assume that after processing an input string x, the circuit system will initiate again and get into a random state, followed by the reading of the first symbol <SOS> of another string y to get into the initial state q_1 for further processing of y.
- Therefore, in the state of q_3, we may assume that no more symbol will be read in by the machine, i.e., a, b and <EOS> will not be the input symbols, so that the transitions $\delta(q_3, a)$, $\delta(q_3, b)$ and $\delta(q_3, \text{<EOS>})$ become the so-called don't-cares in digital systems, i.e., these transitions may be assigned any arbitrary values which can be utilized to simplify the hardware design.
- Another case which can be utilized to simplify circuit design is: the state q_4 can be entered only during the system initialization process, and so we may assume that in q_4, only <SOS> can be read in; any other symbols a, b or <EOS> cannot. Therefore, we may also let $\delta(q_4, a)$, $\delta(q_4, b)$ and $\delta(q_4, \text{<EOS>})$ as don't cares for circuit design simplification.

♦ The process of digital circuit design is as follows.

- Coding of input symbols --- there are four different symbols, a, b, <SOS> and <EOS>; and there need two bits I_1 and I_2 to represent them as follows: $I_2I_1 = 00$ for <SOS>, 01 for a, 10 for b, and 11 for <EOS>.
- Coding of states --- there are four states q_1 through q_4; and there need as well two bits

D_1 and D_2 to represent them as follows: D_2D_1 = 00 for q_4, 01 for q_1, 10 for q_2, 11 for q_3. Note that D_1 and D_2 are also used to specify the flipflops which save the two bit values.

- State transition table of the digital system style --- according to the codings above, we may write down a transition table of the digital system style as shown in Table 4s.5, in which the symbol × is used to represent "don't care."

Table 4s.5 The transition table of the digital system style of Example 4s.7.

Current state D_2D_1	Input x I_2I_1	Next state $D_2'D_1'$	Output y
00	00	01	0
00	01	××	0
00	10	××	0
00	11	××	0
01	00	01	0
01	01	10	0
01	10	01	0
01	11	01	0
10	00	01	0
10	01	10	0
10	10	01	0
10	11	11	0
11	00	01	1
11	01	××	1
11	10	××	1
11	11	××	1

- Boolean expression for the circuit output equation --- from the transition table, we can see that the output of the system depends only on the current state (which comes from the original property of the Mealy machine). Specifically, we can figure out that the Boolean equation for the output y is $y = D_2D_1$.
- Boolean expression for the input equation of the next state D_1' of flipflop D_1 --- this can be derived, by observation, from the Karnaugh map (constructed from the transition table) of D_1 as shown in Fig. 4s.12(a) to be $D_1' = \overline{I}_1 + I_2$ (note: in this process, we have utilized the don't cares marked as × to simplify the resulting expression).
- Boolean expression for the input equation of the next state D_2' of flipflop D_2 --- this can be derived from the Karnaugh map of D_2 as shown in Fig. 4s.12(b) to be $D_2' = \overline{I}_2 I_1 + D_2I_1 = (\overline{I}_2 + D_2)I_1$.
- Digital circuit diagram --- the final digital circuit designed accordingly is shown in Fig. 4s.13.
- Check of the correctness of the designed circuit --- for input string x = <SOS>*baba*<EOS> = (00)(10)(01)(10)(01)(11), we can check to circuit operation to see the output y = ×000001 which is correct, where "×" in y is used to represent the initial random output value of the system before the symbol <SOS> is read in, and the last bit "1" means the input string is accepted.

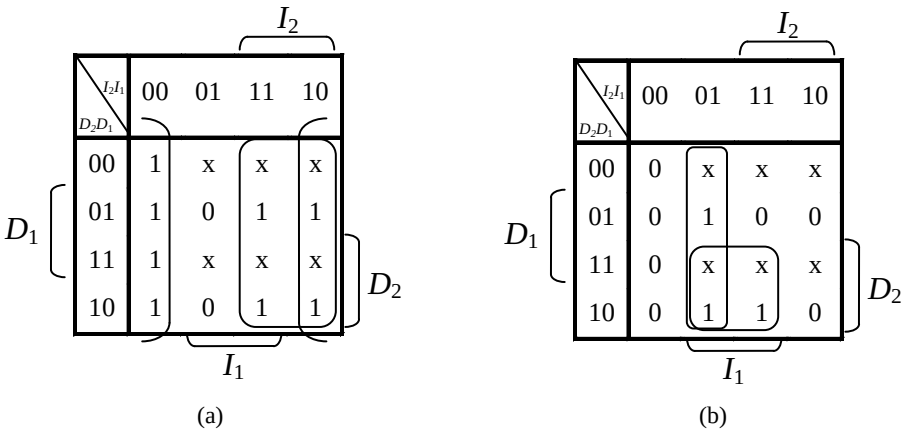


Fig. 4s.12 Karnaugh maps for flipflops D_1 and D_2 of Example 4s.7. (a) Map for D_1'. (b) Map for D_2'.

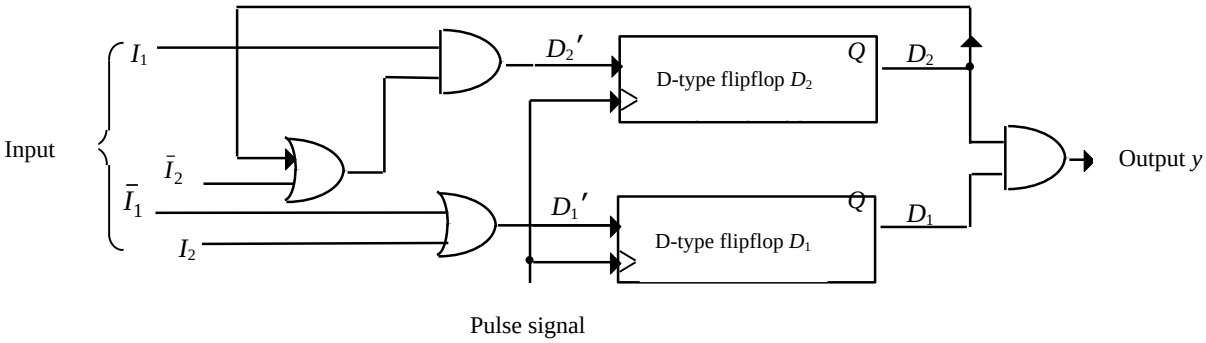


Fig. 4s.13 Designed circuit for Example 4s.7.

4s.5 Equivalence of Mealy Machine and Moore Machine

- **Concepts ---**
  - The Mealy machine and the Moore machine seem to be different in appearance, but actually they can be proven to be equivalent, as will be done in this section.
  - This point in fact have been seen once in Section 4s.3 where we used a Mealy machine and a Moore machine to implement a DFA.
  - It is also noted that with no input symbol, the Moore machine can emit an output symbol while in the initial state.

- **Definition of equivalence of Mealy machine and Moore machine ---**

Given a Mealy machine M_1 and a Moore machine M_2 with respective output functions λ_1 and λ_2, if for all input string x, the equality $b\lambda_1(x) = \lambda_2(x)$ holds where b is the output

of M_2 in its initial state, then M_1 and M_2 are aid to be equivalent.

♦ Note that the equivalence of two SM's based on the equality about strings instead of about languages because we have not defined the language accepted by the SM (Can we do so? Think about it!).

■ **Theorem 4s.1 (Constructing an equivalent Mealy machine from a given Moore machine)** ---

Given a Moore machine $M_1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$, there exists an equivalent Mealy machine M_2.

Proof.

♦ At first, we construct a Mealy machine $M_2 = (Q, \Sigma, \Delta, \delta, \lambda', q_0)$, where λ' is defined in such a way that

"for all $a \in \Sigma$ and $q \in Q$, $\lambda'(q, a) = \lambda(\delta(q, a))$." (A)

♦ With the constructed Mealy machine above, check Fig. 4s.14 to see the relation between λ' and λ.

♦ Now let $b = \lambda(q_0)$. Then, according to the definition of SM equivalence, what we have to do is just to prove the following equality to be true for all $x \in \Sigma^*$:

$$b\lambda'(x) = \lambda(x). \quad \text{(B)}$$

For this, the proof technique of induction with respect to the length $|x|$ of the input string x is adopted as follows.

♦ ***Basis*** ---

- When $|x| = 0$, i.e., when $x = \varepsilon$, the output of the Moore machine is $\lambda(q_0)$ as mentioned before, so $\lambda(x) = \lambda(\varepsilon) = \lambda(q_0) = b$.
- For the input $x = \varepsilon$ which is nothing, the output of the Mealy machine is obviously nothing as well *by definition*, i.e., $\lambda'(q_0, \varepsilon) = \varepsilon$; therefore, $\lambda'(x) = \lambda'(\varepsilon) = \varepsilon$.
- Consequently, we have $b\lambda'(x) = b\varepsilon = b = \lambda(x)$, i.e., for $|x| = 0$, the equality (B) holds.

♦ ***Induction*** ---

- Now suppose that for $|x| = k$ with $x = a_1a_2\ldots a_k$, the equality (B) is true, where $k \geq 0$, which means that the following equality holds:

$$b\lambda'(x) = \lambda(x), \quad \text{(C)}$$

and there is a state sequence $q_1, q_2, \ldots, q_k$ such that $\delta(q_0, a_1) = q_1$, $\delta(q_1, a_2) = q_2, \ldots, \delta(q_{k-1}, a_k) = q_k$.

- When $y = xa_{k+1}$ with $|y| = k + 1$, assume that $\delta(q_k, a_{k+1}) = q_{k+1}$.
- Then, we have

$$\begin{aligned}
\lambda(y) &= \lambda(xa_{k+1}) && (\because y = xa_{k+1}) \\
&= \lambda(x)\lambda(q_{k+1}) && \text{(by definition of } \lambda(xa_{k+1})) \\
&= \lambda(x)\lambda(\delta(q_k, a_{k+1})) && (\because \delta(q_k, a_{k+1}) = q_{k+1}) \\
&= \lambda(x)\lambda'(q_k, a_{k+1}) && \text{(by definition of } \lambda' \text{ in (A))} \\
&= b\lambda'(x)\lambda'(q_k, a_{k+1}) && \text{(according to (C))} \\
&= b\lambda'(xa_{k+1}) && \text{(by definition of } \lambda'(xa_{k+1}))
\end{aligned}$$

$= b\lambda'(y).$ $(\because y = xa_{k+1})$

That is, for $|y| = k + 1$, (B) holds.

- In conclusion, we have proved that for all possible $x \in \Sigma^*$, (B) holds. This completes the proof of the theorem.

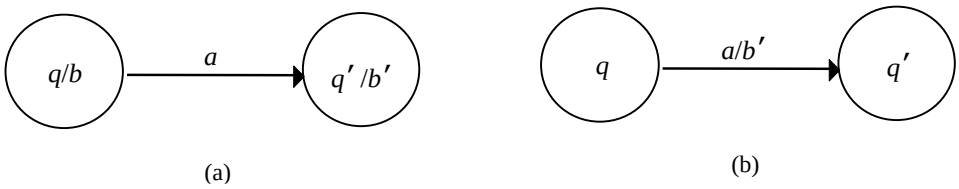
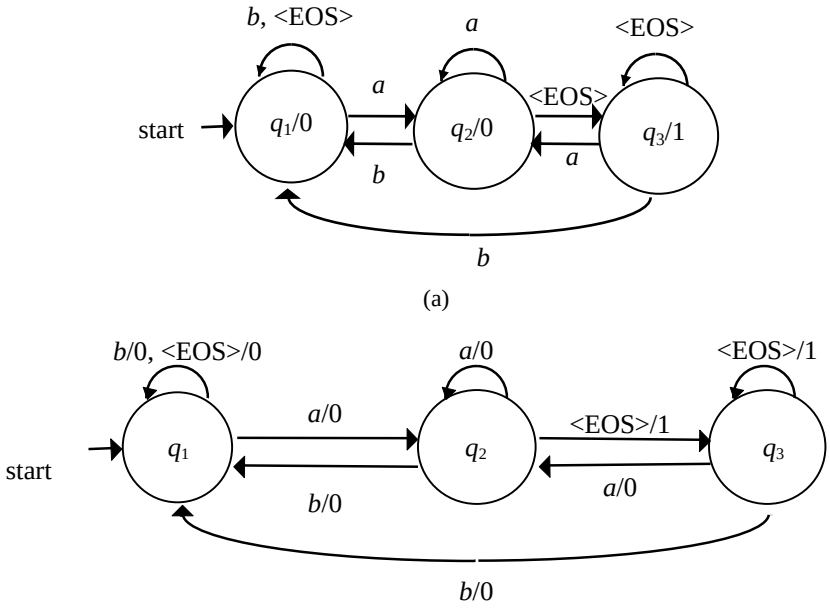


Fig. 4s.14 Constructing an equivalent Mealy machine for a Moore machine. (a) Part of given Moore machine. (b) Part of equivalent Mealy machine.

■ **Example 4s.8** ---

Transform the Moore machine shown in Fig. 4s.10(d) into an equivalent Mealy machine.

- According to Theorem 4s.1, or more specifically, according to the way mentioned in (A) above: "for all $a \in \Sigma$ and $q \in Q$, define $\lambda'(q, a) = \lambda(\delta(q, a))$," a Mealy machine M_2 equivalent to the Moore machine M_1 shown in Fig. 4s.10(d), which is repeated in Fig. 4s.15(a), is shown in Fig. 4s.15(b).
- For example, the transition $\delta(q_2, b) = q_1$ and the output $\lambda(q_1) = 0$ of M_1 leads to creation of the output $\lambda'(q_2, b) = 0$ of M_2.
- The Mealy machine shown in Fig. 4s.15 actually can be simplified to be that shown in Fig. 4s.10(c).
- Simplification of SM's will be discussed later in the next section.



(b)

Fig. 4s.15 Transforming a Moore machine into an equivalent Mealy machine. (a) Given Moore machine M_1 (the same as that shown in Fig. 4s.10(c)). (b) The obtained Mealy machine M_2 equivalent to that in (a).

■ **Theorem 4s.2 (Constructing an equivalent Moore machine from a given Mealy machine) ---**

Given a Mealy machine $M_1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$, there exists an equivalent Moore machine M_2.

Proof.

♦ At first, we construct a Moore machine $M_2 = (Q, \Sigma, \Delta, \delta, \lambda', q_0')$ where

1. $Q' = Q \times \Delta = \{[q, b] \mid q \in Q, b \in \Delta\}$;
2. for all $[q, b] \in Q'$ and $a \in \Sigma$, define

$$\delta'([q, b], a) = [\delta(q, a), \lambda(q, a)] \text{ and } \lambda'([q, b]) = b; \quad \text{(A)}$$

3. $q_0' = [q_0, b_0]$, where b_0 is any element selected from Δ.

♦ The relation between δ', λ' and δ, λ can be seen in Fig. 4s.16.

♦ Now let $b_0 = \lambda'(q_0') = \lambda'([q_0, b_0])$. Now let $b = \lambda(q_0)$. Then, according to the definition of SM equivalence, what we have to do is just to prove the following equality to be true for all $x \in \Sigma^*$:

$$b_0\lambda(x) = \lambda'(x), \quad \text{(B)}$$

For this, the proof technique of induction with respect to the length $|x|$ of the input string x is adopted as follows.

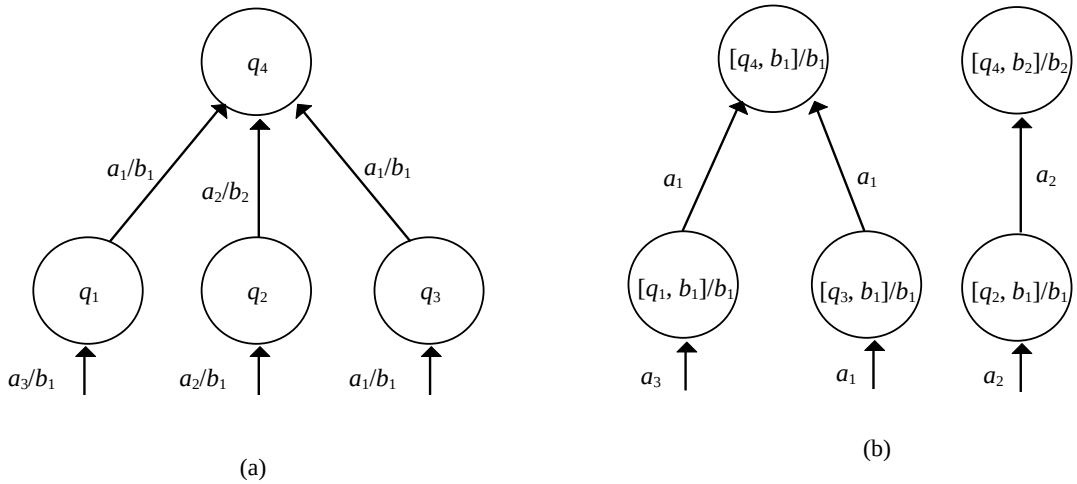


Fig. 4s.16 Constructing an equivalent Moore machine for a Mealy machine. (a) Part of given Mealy machine. (b) Part of equivalent Moore machine.

♦ ***Basis*** **---**

- When $|x| = 0$, i.e., when $x = \varepsilon$, the output of Moore machine is $\lambda'(q_0')$, so $\lambda'(x) = \lambda'(\varepsilon) = \lambda'(q_0') = \lambda'([q_0, b_0]) = b_0$.
- For the input ε, the output of Mealy machine is ε, i.e., $\lambda(q_0, \varepsilon) = \varepsilon$ as defined; so $\lambda(x) = \lambda(\varepsilon) = \varepsilon$.
- Consequently, we have $b_0\lambda(x) = b_0\varepsilon = b_0 = \lambda'(x)$, i.e., for $|x| = 0$, (B) holds.

♦ ***Induction*** ---

- Now suppose that for $|x| = k$ with $x = a_1a_2\ldots a_k$, the equality (B) is true, where $k \geq 0$, which means that the following equality holds:

$$b_0\lambda(x) = \lambda'(x) \qquad \text{(C)}$$

and there is a state sequence $q_1, q_2, \ldots, q_k$ such that

1. $\delta(q_0, a_1) = q_1$, $\delta(q_1, a_2) = q_2, \ldots, \delta(q_{k-1}, a_k) = q_k$;
2. $\lambda(x) = \lambda(a_1a_2\ldots a_k) = \lambda(q_0, a_1)\lambda(q_1, a_2)\ldots\lambda(q_{k-1}, a_k) = b_1b_2\ldots b_k$
   (assume that $b_i = \lambda(q_{i-1}, a_i)$ here);
3. $\delta'(q_0', a_1) = \delta'([q_0, b_0], a_1) = [\delta(q_0, a_1), \lambda(q_0, a_1)] = [q_1, b_1] = q_1'$,
   $\delta'(q_1', a_2) = \delta'([q_1, b_1], a_2) = [\delta(q_1, a_2), \lambda(q_1, a_2)] = [q_2, b_2] = q_2'$,
   …,
   $\delta'(q_{k-1}', a_k) = \delta'([q_{k-1}, b_{k-1}], a_k) = [\delta(q_{k-1}, a_k), \lambda(q_{k-1}, a_k)] = [q_k, b_k] = q_k'$
   (according to (A) and assume that $q_i' = [q_i, b_i]$);
4. $\lambda'(x) = \lambda'(a_1a_2\ldots a_k) = \lambda'(q_0')\,\lambda'(q_1')\ldots\lambda'(q_k') = \lambda'([q_0, b_0])\lambda'([q_1, b_1])\ldots\lambda'([q_k, b_k])$
   $= b_0b_1\ldots b_k$
   (according to (A)).

- When $y = xa_{k+1}$, assume that $\delta(q_k, a_{k+1}) = q_{k+1}$, $\lambda(q_k, a_{k+1}) = b_{k+1}$, then according to (A), we have

$$\delta'(q_k', a_{k+1}) = \delta'([q_k, b_k], a_{k+1}) = [\delta(q_k, a_{k+1}), \lambda(q_k, a_{k+1})] = [q_{k+1}, b_{k+1}] = q_{k+1}'$$

(assume that $[q_{k+1}, b_{k+1}] = q_{k+1}'$),

and

$$\lambda'(q_{k+1}') = \lambda'([q_{k+1}, b_{k+1}]) = b_{k+1}.$$

- Also,

$$\begin{aligned}
b_0\lambda(y) &= b_0\lambda(xa_{k+1}) && (\because y = xa_{k+1})\\
&= b_0\lambda(x)\lambda(q_k, a_{k+1}) && \text{(according to definition of } \lambda(xa_{k+1}))\\
&= b_0\lambda(x)b_{k+1} && (\because \lambda(q_k, a_{k+1}) = b_{k+1})\\
&= b_0\lambda(x)\lambda'(q_{k+1}') && (\because \lambda'(q_{k+1}') = \lambda'([q_{k+1}, b_{k+1}]) = b_{k+1})\\
&= \lambda'(x)\,\lambda'(q_{k+1}') && \text{(according to (C))}\\
&= \lambda'(xa_{k+1}) && \text{(according to definition of } \lambda'(xa_{k+1}))\\
&= \lambda'(y). && (\because y = xa_{k+1})
\end{aligned}$$

That is, for $|y| = k + 1$, (B) holds.

- In conclusion, we have proved that for all possible $x \in \Sigma^*$, (B) holds. This completes the proof of the theorem.

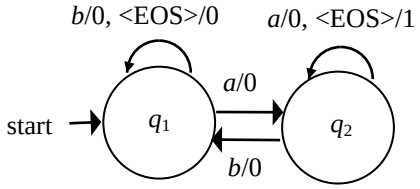
■ **Example 4s.9** ---

Transform the Mealy machine shown in Fig. 4s.10(c) into an equivalent Moore machine.

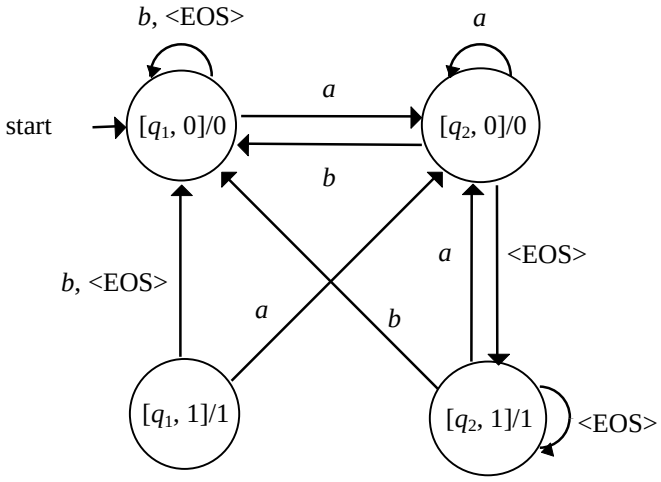
♦ According to Theorem 4s.2, or more specifically, according to the way mentioned in (A) above: "for all $[q, b] \in Q'$ and $a \in \Sigma$, define $\delta'([q, b], a) = [\delta(q, a), \lambda(q, a)]$ and $\lambda'([q, b]) = b$," a Moore machine M_2 equivalent to the Mealy machine M_1 shown in Fig. 4s.10(c), which is repeated in Fig. 4s.16(a), is shown in Fig. 4s.16(b).

♦ For example, the transition $\delta(q_2, b) = q_1$ and the output $\lambda(q_2, b) = 0$ of M_1 leads to creation of the transition $\delta'([q_2, 0], b) = [\delta(q_2, b), \lambda(q_2, b)] = [q_1, 0]$ and output $\lambda'([q_2, 0]) = 0$ of M_2.

♦ The initial state of the constructed Moore machine is $[q_1, 0]$, and $[q_1, 1]$ is obviously inaccessible and so can be deleted. As can be seen, the resulting machine becomes identical to that of Fig. 4s.10(d) in structure.



(a)



(b)

Fig. 4s.16 Transforming a Mealy machine into an equivalent Moore machine. (a) Given Mealy machine M_1 (same as that shown in Fig. 4s.10(c)). (b) The obtained Mealy machine M_2 equivalent to that in (a).

4s.6 Simplification of Sequential Machines

- Like the DFA, the SM also can be simplified after removing the inaccessible states from the initial state.
- A difference here is that to differentiate two states, no state cannot be used; instead, the output strings are utilized.
- Before getting into the details, some definitions of notations and terms are given at first as follows.
  - Given a Mealy machine $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$, for state $p \in Q$ and input string $x = a_1a_2\ldots a_n$, if $\delta(p, a_1) = p_1$, $\delta(p_1, a_2) = p_2$, …, $\delta(p_{n-1}, a_n) = p_n$, we define

    $$\lambda_p(x) = \lambda(p, a_1)\lambda(p_1, a_2)\ldots\lambda(p_{n-1}, a_n).$$

    - Note that $\lambda_{q_0}(x) = \lambda(x)$.
  - Then, for any $p, q \in Q$, if there exists a path $x = a_1a_2\ldots a_n$ such that $\lambda_p(x) \neq \lambda_q(x)$, then we say that p and q are *differentiable* by x.
  - If two states are differentiable by a string x with length smaller than or equal to k, then we say that p and q are *k-differentiable* and indicate this by the notation $p \overset{k}{\equiv} q$; otherwise, we say that p and q are k-indifferentiable.

- The concept of k- indifferentiability defines a kind of equivalence relation $\overset{k}{\equiv}$ by which *all the states of a Mealy machine* may be divided into several *equivalence classes* and the states in each equivalence class are all k-indifferentiable.
  - It can be figured out that when $k = 0$, i.e., when the input string x is ε, there is only one equivalence class, which is just Q itself; and when $k = 1$, i.e., when x includes only one symbol, the equivalence relation $\overset{1}{\equiv}$ will divide all the states into several classes.
    - In more detail, if for all symbol $a \in \Sigma$ and states p and q, $\lambda(p, a) = \lambda(q, a)$ is true, then p and q are *1-indifferentable*.
    - Therefore, to partition Q into 1-indefferentiable equivalence classes, the scheme is to find states from which for all input symbols, the output symbols are identical, and put such states into an identical equivalence class.
    - Let the result of such a state division scheme, which is a set of 1-indefferentiable equivalence classes, be denoted as P_1.

  - Then, we use the equivalence relation $\overset{2}{\equiv}$ to divide P_1 further.
    - It can be figured out that if p and q are 1-indifferentiable, and for all $a \in \Sigma$, $\delta(p, a)$ and $\delta(q, a)$ are both 1-indifferentiable, then p and q are *2-indifferntiable*.
    - Therefore, the scheme to divide P_1 into 2-indifferentiable equivalence classes is:

      if p and q are in an identical equivalence class, and $\delta(p, a)$ and $q(p, a)$ are also in an identical equivalence class, then put p and q into an identical equivalence class.

    - The above scheme will result in a set of 2-indifferentiable equivalence classes, which we denote as P_2.

  - The above process can be performed recursively k times to result in a set of k-indifferentiable equivalence classes, which we denote as P_k.
  - As a summary, in general, the following scheme is conducted:

    if p and q are in an identical k-indifferentiable equivalence class of the set P_k,

and $\delta(p, a)$ and $\delta(q, a)$ are in an identical class in P_k, then put p and q into an identical (k+1)-indifferentiable equivalence class in the set P_{k+1}.

- ♦ The above scheme is applied iteratively until all the states in each equivalence class are indifferentiable, and then the states in each class are merged into a single state and renamed, resulting in a *minimum* Mealy machine.

- ■ The above process is for minimizing the Mealy machine. For the minimizing the Moore machine, the process is similar and is omitted.
- ■ The above process may be written as an algorithm, and proved to be correct in a way similar to that for proving the correctness of the process of minimizing the finite automaton. The details are left to the reader.

- ■ **Example 4s.10** ---

  Minimize the Mealy machine whose state transition table is shown in Table 4s.6.

  - ♦ According to the above process, the minimization result is shown in Fig. 4s.17.
  - ♦ The intermediate results are shown in the following, where we use a pair of parentheses to enclose an indifferentiable equivalence class.

    - $P_0 = \{(A, B, C, D, E, F)\}$ ($\because$ all states are 0-indifferentiable);
    - $P_1 = \{(A, C, E), (B, D, F)\}$
      ($\because \lambda(A, 1) = \lambda(C, 1) = \lambda(E, 1) = 1$ and $\lambda(B, 1) = \lambda(D, 1) = \lambda(F, 1) = 0$);
    - $P_2 = \{(A, C, E), (B, D), (F)\}$
      ($\because \delta(D, 1) = B$ and $\delta(F, 1) = C$, but B and C are in different classes in P_1, so D and F are not in an identical class in P_2);
    - $P_3 = \{(A, C), (E), (B, D), (F)\}$
      ($\because \delta(C, 1) = B$ and $\delta(E, 1) = F$, but B and F are in different classes in P_2, so C and E are not in an identical class in P_3);
    - $P_4 = \{(A, C), (E), (B, D), (F)\} = P_3$; therefore, the process is stopped, and the result is: A and C are indifferentiable, and B and D are indifferentiable;
    - Rename the merged A and C by $[A, C]$, the merged B and D by $[B, D]$, and the remaining state E by $[E]$, state F by $[F]$.

  - ♦ The final result is as shown in Table 4s.7.

Table 4s.6 Given Mealy machine of Example 4s.10.

input / current state	0	1
A	$E/0$	$D/1$
B	$F/0$	$D/0$
C	$E/0$	$B/1$
D	$F/0$	$B/0$
E	$C/0$	$F/1$
F	$B/0$	$C/0$

Table 4s.7 Minimized version of Mealy machine of Table 4s.6.

input / current state	0	1
$[A, C]$	$[E]/0$	$[B, D]/1$
$[B, D]$	$[F]/0$	$[B, D]/0$
$[E]$	$[A, C]/0$	$[F]/1$
$[F]$	$[B, D]/0$	$[A, C]/0$