Chapter 2

Finite Automata

(part A)

(2015/10/20)

Hokkaido, Japan

Outline

2.0 Introduction

- Two types of finite automata (FA):
  - deterministic FA (DFA)
  - nondeterministic FA (NFA)

- Both types are of the same power:
  - the former is easier for hardware and software implementations; and
  - the latter is more efficient for descriptions of applications of FA.

2.1 An Informal Picture of Finite Automata

- A complete application example of finite automata for protocol design and verification
  - Involving a concept of “product of two automata” (read details in the textbook by yourself)

2.2 Deterministic Finite Automata

2.2.0 A Review (supplemental)

- Recall the example of designing a vending machine selling 20-dolllar food packs in Chapter 0 (see Fig. 2.1).

- What abstract concepts are involved in the design? ⇒ See the definition next.

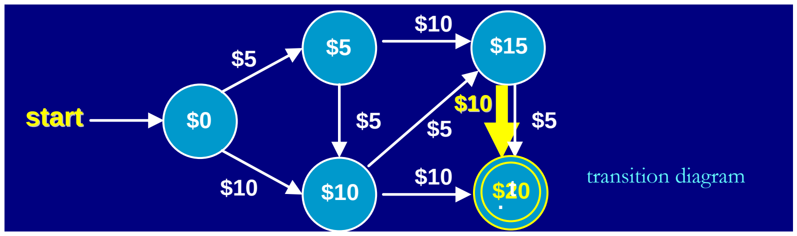

Fig. 2.1 Transition diagram of a finite automaton for a vending machine (from Chapter 0).

2.2.1 Definition of DFA

- Definition ---

  A *deterministic finite automata* (DFA) *A* consists of 5-tuples (1/2):
  - a finite (nonempty) set of *states* Q;
  - a finite (nonempty) set of *input symbols* Σ;
  - a (state) *transition function* δ such that

$$\delta(q, a) = p$$

  Means “automaton *A*, in state *q*, takes input *a* and enters state *p*.”
  - a *start state* q_0;
  - a set of (nonempty) *final or accepting states* F.

- DFA A may be written as a "5-tuple" as follows:

  $$A = (Q, \Sigma, \delta, q_0, F).$$

- A graphic model for a DFA --- see Fig. 2.2.

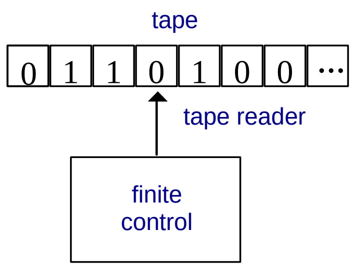

Fig. 2.2 A graphic model for a DFA.

2.2.2 <u>How a DFA processes strings</u>

- Given an input string $x = a_1a_2\ldots a_n$, if

  $$\delta(q_0, a_1) = q_1, \delta(q_1, a_2) = q_2, \ldots, \delta(q_{i-1}, a_i) = q_i, ..., \delta(q_{n-1}, a_n) = q_n, \text{ and } q_n \in F,$$

  then x is "accepted"; otherwise, "rejected."

  ♦ Every transition is *deterministic*.

- Example 2.1 ---

  Design an FA A to accept the language

  $$L = \{x01y \mid x \text{ and } y \text{ are any strings of 0's and 1's}\}.$$

  ♦ Examples of strings in L: <u>01</u>, 11<u>01</u>0, 100<u>01</u>1…

  ♦ Transitions:

  $$\delta(q_0, 1) = q_0, \delta(q_0, 0) = q_2, \delta(q_2, 1) = q_1, \delta(q_2, 0) = q_2, \delta(q_1, 0) = q_1, \delta(q_1, 1) = q_1$$

  ♦ 5-Tuple:

  $$A = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_1\})$$

2.2.3 <u>Simpler Notations for DFA's</u>

- For the last example, we may have other ways of descriptions as follows.

  ♦ *Transition diagram* --- see Fig. 2.3.

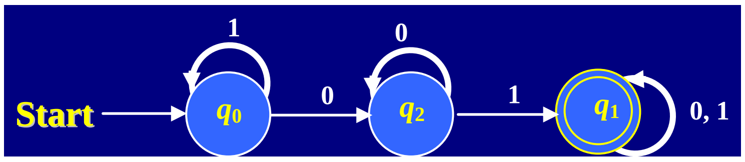

Fig. 2.3 The transition diagram for the DFA of Example 2.1.

♦ *Transition table* --- see Table 2.1.

Table 2.1 The transition table for Example 2.1.

	0	1
$\rightarrow q_0$	q_2	q_0
$*q_1$	q_1	q_1
q_2	q_2	q_1

(Notations: →: initial state; *: final state)

2.2.4 <u>Extending transition function to strings</u>

- *Extended transition function* $\hat{\delta}$ ---

  If $x = a_1a_2\ldots a_n$ and δ is such that

$$\delta(p, a_1) = q_1,\ \delta(q_1, a_2) = q_2,\ \ldots,\ \delta(q_{i-1}, a_i) = q_i,\ \ldots,\ \delta(q_{n-1}, a_n) = q,$$

  then we define $\hat{\delta}$ to be $\hat{\delta}(p, x) = q$.

- $\hat{\delta}$ may also be defined recursively as follows:

  Basis: $\hat{\delta}(q, \varepsilon) = q$.

  Induction: if $w = xa$ (a is the last symbol of w), then $\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a)$.

- A graphic diagram for the concept of the above induction step --- see Fig. 2.4.

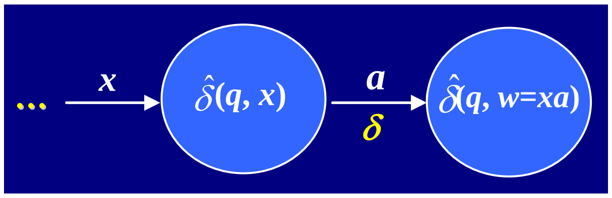

Fig. 2.4 An illustration for the induction step in the definition of $\hat{\delta}$.

- Difference between δ and $\hat{\delta}$ --- the former accepts single symbols as input while the latter may accept strings as input.

2.2.5 <u>The Language of a DFA</u>

- The language of a DFA A is defined as

$$L(A) = \{w \mid \hat{\delta}(q_0, w) \text{ is in } F\}.$$

- If L is $L(A)$ for some DFA A, then we say L is a *regular language*.

2.3 Nondeterministic Finite Automata

2.3.1 An informal view of NFA's

■ Review of a previous example of DFA (of Example 2.1) ---

♦ Original version --- see Fig. 2.5 (the same as Fig. 2.3)

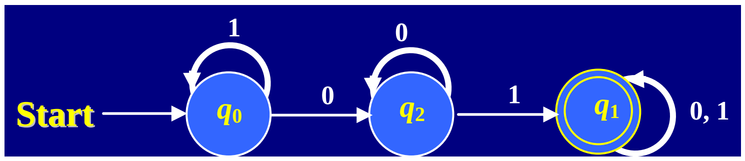

Fig. 2.5 The transition diagram for the DFA of Example 2.1 (Fig. 2.3 repeated here for comparison).

♦ A nondeterministic finite automaton (NFA) version of the above DFA --- see Fig. 2.6.

- More intuitive!
- How to design an NFA will be described later.

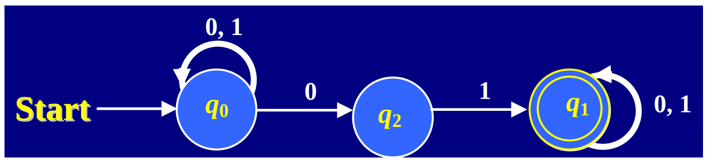

Fig. 2.6 Another transition diagram for the DFA of Example 2.1 using the extended transition function $\hat{\delta}$.

■ Some properties of NFA's (see Fig. 2.6 for the illustration) ---

♦ Some transitions may "die," like $\hat{\delta}(q_2, 0)$.

♦ Some transitions have multiple choices, like $\hat{\delta}(q_0, 0) = q_0$ and q_2.

■ Example 2.6 ---

Design an NFA accepting the following language

$$L = \{w \mid w \in \{0, 1\}^* \text{ and ends in } 01\}.$$

♦ By *intuition*, we can draw the transition diagram for an NFA A to accept L as shown in Fig. 2.7.

♦ Nondeterminism may create many transition paths, but if there is one path leading to a final state, then the input is accepted.

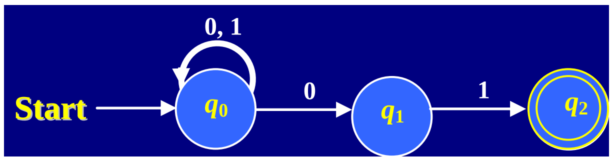

Fig. 2.7 A design of the NFA of Example 2.2 by intuition.

♦ When input x = 00101, the NFA processes x in a way as shown in Fig. 2.8.

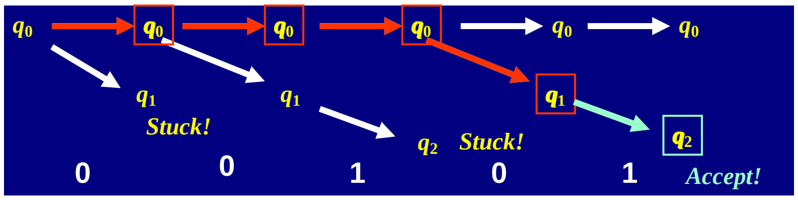

Fig. 2.8 The process of the NFA of Example 2.2 to accept the input x = 00101 (Fig. 2.10 in the textbook).

2.3.2 Definition of NFA

■ An NFA A is a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$ where
- ♦ Q = a finite (nonempty) set of states;
- ♦ Σ = a finite (nonempty) set of input symbols;
- ♦ q_0 = a start state;
- ♦ F = a set of (nonempty) final or accepting states;
- ♦ δ = a (state) transition function such that

$$\delta(q, a) = \{p_1, p_2, \ldots, p_m\}$$

which means "automaton A, in state q, takes input a and enters one of the states p_1, p_2, …, p_m."

■ Example 2.7 ---
The transition table of the NFA of the last example (Example 2.6) is as shown in Table 2.2.

Table 2.2 Transition table of NFA of Example 2.2.

	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	ϕ	$\{q_2\}$
$*q_2$	ϕ	ϕ

2.3.3 Extended Transition Function

■ *Recursive* definition of $\hat{\delta}$ (with strings, not symbols, as input) ---

Basis: $\hat{\delta}(q, \varepsilon) = \{q\}$.

Induction: if $w = xa$ (with a as the last symbol of w), $\hat{\delta}(q, x) = \{p_1, p_2, \ldots, p_k\}$, and

$$\delta(p_i, a) = \{r_1, r_2, \ldots, r_m\}, \text{ then } \hat{\delta}(q, w) = \{r_1, r_2, \ldots, r_m\}.$$

■ A graphic diagram for the concept involved in the induction step above is as shown in Fig. 2.9(supplemental):

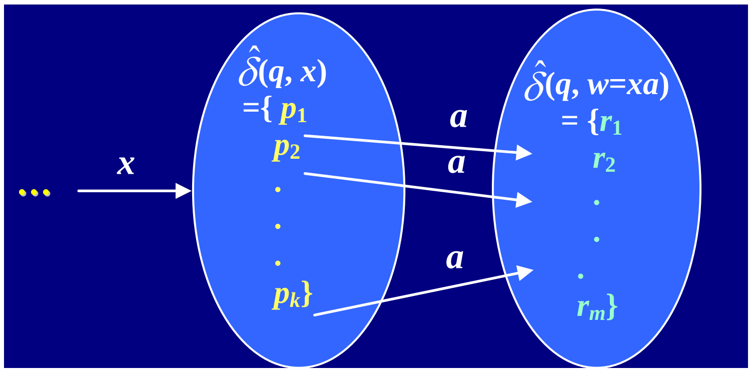

Fig. 2.10 A graphic model for explaining the induction step of the extended transition function.

- Example 2.8 (continued from Example 2.6) ---

  With the input w = 00101 for the NFA of Example 2.6, we have

  1. $\hat{\delta}(q_0, \varepsilon) = \{q_0\}$;
  2. $\hat{\delta}(q_0, \varepsilon 0) = \delta(q_0, 0) = \{q_0, q_1\}$;
  3. $\hat{\delta}(q_0, 00) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \phi = \{q_0, q_1\}\ldots$

2.3.4 The Language of an NFA

- Definition ---

  If $A = (Q, \Sigma, \delta, q_0, F)$ is an NFA, then the language accepted by A is

  $$L(A) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \phi\}.$$

  - That is, as long as a final state is reached, which is among possibly many, the input is accepted.

2.3.5 Equivalence of DFA and NFA

- A comparison of DFA's and NFA's is as follows.
  - NFA's are more intuitive to construct for many languages;
  - DFA's are easier for use in software and hardware implementations.

- Definition: "Two models are said to be equivalent if they have identical languages."

- Theorem: Every NFA N has an *equivalent* DFA D, i.e., $L(D) = L(N)$.
  - The proof of the theorem is based on the technique of "subset construction."
  - Concept of subset construction: each state of DFA is a subset of NFA
  - Detail of proof:

    Given an NFA $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$, we construct a DFA $D = (Q_D, \Sigma, \delta_D, q_0', F_D)$ such that:
    - the two alphabets are identical;
    - Q_D is the power set of Q_N (i.e., set of all subsets of Q_N);
    - inaccessible states in Q_D are deleted;
    - $q_0' = \{q_0\}$;
    - each subset S of Q_N is put in the final state set F_D if S includes at least one accepting

state in F_N (i.e., $S \cap F_N \neq \phi$);

- for each subset S of Q_N and for each input symbol a, δ_D is defined as

$$\delta_D(S, a) = \bigcup_{p \in S} \delta_N(p, a).$$

■ Example 2.10 ---

For the NFA of Example 2.6 (see Fig. 2.7), we have

♦ The power set of $Q_N = \{q_0, q_1, q_2\}$ is $Q_D = \{\phi, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$.

♦ The transition table is as shown Table 2.3.

Table 2.3 The transition table for the NFA of Example 2.6.

	0	1
ϕ	ϕ	ϕ
$\rightarrow\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	ϕ	$\{q_2\}$
$*\{q_2\}$	ϕ	ϕ
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$*\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$*\{q_1, q_2\}$	ϕ	$\{q_2\}$
$*\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

♦ Elimination of *inaccessible states* ---

- Changing the names of the states in Table 2.3 according to the following rules, we get a result as shown in Table 2.4 (which is Fig. 2.13 in the textbook):

ϕ = A, $\{q_0\}$ = B, $\{q_1\}$ = C, $\{q_2\}$ = D, $\{q_0, q_1\}$ = E,
$\{q_0, q_2\}$ = F, $\{q_1, q_2\}$ = G, $\{q_0, q_1, q_2\}$ = H.

Table 2.4 A transition table transformed from Table 2.3 for the NFA of Example 2.6.

	0	1
A	A	A
→B	E	B
C	A	D
*D	A	A
E	E	F
*F	E	B
*G	A	D
*H	E	F

- Checking *accessible states* from the start state (marked by "→,"), we get a result shown as the red ones in Table 2.4 (only B, E, F left); the other states are *inaccessible*.
- A transition diagram for the above is shown in Fig. 2.11.

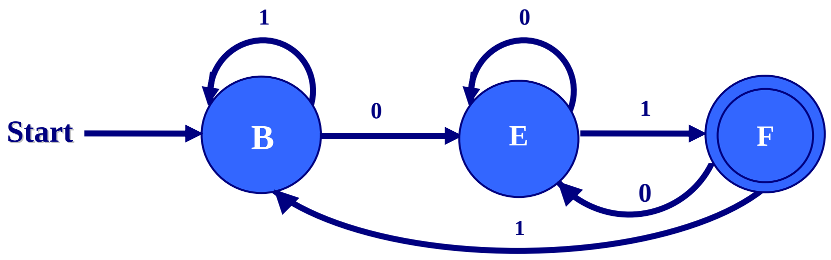

Fig. 2.11 The transition diagram for the simplified NFA of Example 2.10.

■ Another way to eliminate inaccessible states by "lazy evaluation" ---
starting from the start state, *only accessible states* are evaluated in order to avoid generation of inaccessible states (see Fig. 2.7 again during the evaluation process).

■ Example 2.10 (continued) ---

♦ The result according to lazy evaluation for Example 2.10 using Table 2.3 is shown by the red parts in Table 2.5 which are the same as those red states shown in Table 2.4.

♦ The corresponding transition diagram is shown in Fig. 2.12.

Table 2.5 The transition table for the NFA of Example 2.6 using "lazy evaluation."

	0	1
ϕ	ϕ	ϕ
$\rightarrow\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	ϕ	$\{q_2\}$
$*\{q_2\}$	ϕ	ϕ
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$*\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$*\{q_1, q_2\}$	ϕ	$\{q_2\}$
$*\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

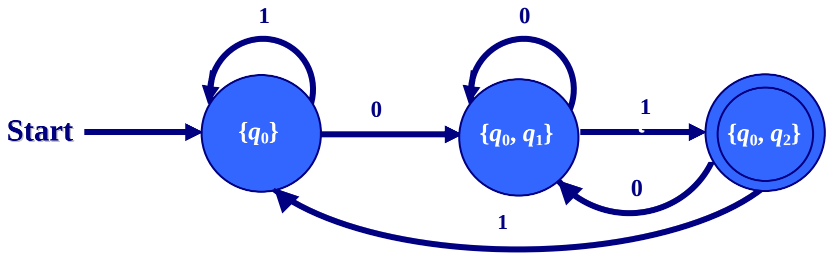

Fig. 2.12 The transition diagram of Example 2.10 resulting from lazy evaluation.

■ *Theorem 2.11* —

If DFA $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ is constructed from NFA $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$ by subset construction, then $L(D) = L(N)$.

♦ Proof. See the textbook.

■ *Theorem 2.12 (equivalence of DFA and NFA)*:

A language L is accepted by some DFA *if and only if* L is accepted by some NFA.

♦ Proof. See the textbook.

2.3.6 A Bad Case of Subset Construction

■ The DFA *constructed* from an NFA usually has the same number of accessible states of the NFA. But the worse case is $m = 2^n$ where
- ♦ m is the number of states in the DFA.
- ♦ n is the number of states in the NFA.

■ Regarding NFA's as DFA's ---
- ♦ In each state of a DFA, for each input there must be a next state.
- ♦ In an NFA, for a certain input there might be no next state. So the automaton shown in Fig. 2.13 is an NFA.
- ♦ But it is deterministic in nature --- either getting into a next state or becoming "dead" (getting into a "dead" state).
- ♦ Such a kind of NFA has at most one transition out of any state on any symbol, and may be regarded as a DFA.

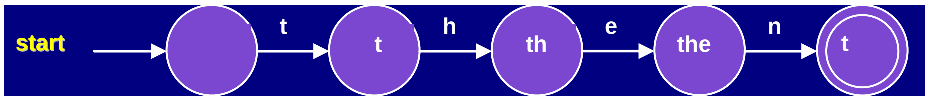

Fig. 2.13 An NFA which may be regarded as a DFA.

(to be continued in part II of this chapter)