4CS4-6: Theory of Computation (FA, Reg. Expr., Tran. graph, Tran. matrix)

Lecture 02: Jan. 09, 2020

Prof. K.R. Chowdhary *: Professor of CS*

2.1 Models of FA

The conceptual model of finite-state machine is illustrated in the figure 2.1. The strings of symbols are read (input) to the FA through an *input tape*, which has equal size squares on it, and each of them holds one symbol. The principle part of FA is *finite control*, which occupies in one of the states at a time out of a number of states. The finite control has *reading head*, which always points to the input tape and can read one symbol at a time. There is no writing head, hence FA can only **read** the input symbols from the input tape.

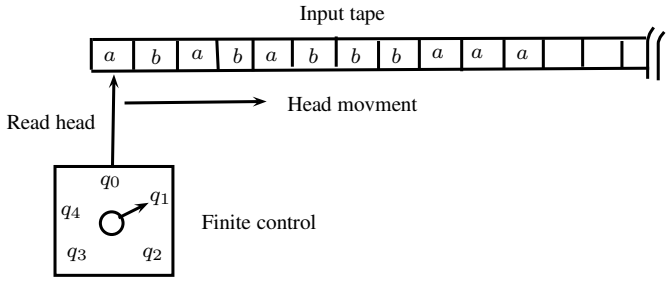

Figure 2.1: Finite Automata.

Before reading of input starts, the read head remains pointing to the left most square on the tape and the finite control is set to the specified initial state called *start state*. Every time a symbol is read from the tape, the finite control moves to a definite next state depending on symbol read and the current state of the finite control, and the read head advances to next-square on the input tape. This process continues until the input symbol's sting on the tape is exhausted, i.e., end of the input string is reached. Thus, the new or next state is function of two parameters: (i) the current state of FA, and (ii) the current input symbol at the read head. Because the next state is definite or deterministic in nature, the automaton is called *Deterministic Finite Automaton* (DFA).

When FA ultimately reaches to the end of the input string, it indicates its approval or disapproval about the acceptance of the input string through the state in which it finally lands. If it is one of the final states, the input string is accepted otherwise not. In this way a finite automata ultimately leads to final state for certain (valid) input strings, and to

non-final for other strings. Thus, language accepted by a particular FA is the set of strings that it accepts.

2.1.1 Mathematical Model

After above discussion, we are in a position to formally define the FA, called DFA, as a 5-tuple. We shall represent the FA by symbol M, such that,

$$M = (Q, \Sigma, \delta, s, F) \tag{2.1}$$

where,

Σ is a finite set of input alphabet (every symbol present on input tape is in Σ),

Q is set of *finite states*,

δ is *transition function*, which decides new state based on the current state and current input symbol. Therefore,

$$\delta : Q \times \Sigma \mapsto Q \tag{2.2}$$

s is *start state* of DFA, $s \in Q$, and

F is set of *final* or *accepting states*, and $F \subseteq Q$

The rules according to which finite automaton M picks up the next state are encoded in the transition function δ (delta). Therefore, if DFA M is in state $q \in Q$ and the symbol read from input tape is $\sigma \in \Sigma$, then next state is $\delta(q, \sigma) \in Q$, which is uniquely determined state to which M will move.

Given an input string, the DFA will go through different *configurations* when it reads the input symbols one after another. Each configuration (also called ID - *Instantaneous Description*) of DFA is combination of - *finite control* and *input tape*. In other words, each configuration is an element of $Q \times \Sigma^*$. For a configuration $(q_i, abcd)$, where current state is q_i and input symbol a, if the DFA moves to new state q_j then its new configuration will be (q_j, bcd). A binary relation $\vdash_M$ (read as derives) represents this change of configuration in a single move. Hence,

$$(q_i, abcd) \vdash_M (q_j, bcd) \tag{2.3}$$

where $a, b, c, d \in \Sigma$ and $q_i, q_j \in Q$. Figure 2.2(a) and (b) show two different configurations of a DFA.

In general, we can say (q_i, u) yields (q_j, v) in single move specified as $q_i u \vdash q_j v$, where $u, v \in \Sigma^*$. In the process of movement of DFA from one configuration to next, ultimately v becomes *null* when end of the input string is reached. After this, no further move takes place. Hence, $\vdash_M$ is a mapping from $Q \times \Sigma^*$ to $Q \times \Sigma^*$ expressed as,

$$\vdash_M : Q \times \Sigma^* \mapsto Q \times \Sigma^* \tag{2.4}$$

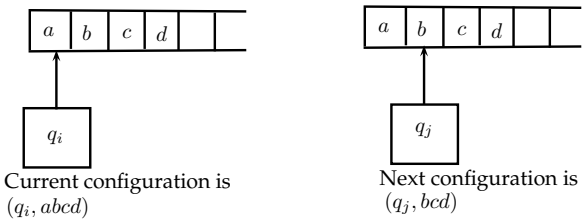

Figure 2.2: (a) Read-head at start, (b) Read-head at intermediate position.

2.1.2 FA as Language Recognizer

Let us ponder what problems this simple machine can solve. Most problems in mathematics require memory, and the size of memory required is arbitrary for arbitrary size of the problem. One of the problems in this category is of primality testing. As the number to be tested for primality becomes larger and larger, the memory required becomes larger and larger. On the other hand is the problem to test whether a given sequence of binary bits has odd or even parity. This problem will require finite memory and arbitrary size of the tape to hold input string of binary sequence. The finite automata is considered fit for solving this second category of problems.

Consider that a configuration (q, w) results to (q', w') the configuration after certain moves have taken place by the FA as,

$$(q, w) \vdash_M (q_1, r) \vdash_M (q_2, s) \vdash_M \cdots \vdash_M (q', w'). \tag{2.5}$$

This can be represented by $(q, w) \vdash_M^* (q', w')$, where $\vdash_M^*$ is called *transitive closure* of $\vdash_M$, showing that FA moves from configuration (q, w) to (q', w') in zero or more number of moves. The string $w \in \Sigma^*$ is accepted by the finite automata if $w' = \varepsilon$ (null string) and $q' \in F$. Thus, the language accepted by the DFA M, represented by $L(M)$ is set of all the strings accepted by M, i.e.,

$$L(M) = \{w \mid w \in \Sigma^* \text{ and } (q_0, w) \vdash_M^* (q', \varepsilon) \text{ and } q' \in F\}. \tag{2.6}$$

Example 2.1 *Represent the DFA $M = (Q, \Sigma, \delta, s, F)$ using transition graph, and find out the language recognized by this.*

$Q = \{q_0, q_1, q_2\}$

$\Sigma = \{0, 1\}$

$s = q_0$

$F = \{q_1\}$

δ is transition table 2.1.

The transition graph (also called *state diagram*) for M, corresponding to the transition table 2.1 is shown in figure 2.3. The state q_0 is starting state, shown by a circle with

Table 2.1: δ functional relationship.

Current state q	Next state for input 0	Next state for input 1
q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_2	q_1

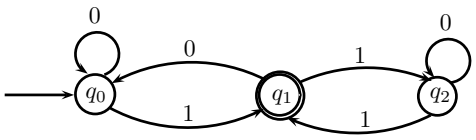

Figure 2.3: Finite Automata example.

incoming arrow. Final state is enclosed in double circles to make it distinct from other states.

If M is given an input of 00110001, the various configurations to which the machine goes through are:

$$\begin{aligned}(q_0, 00110001) &\vdash_M (q_0, 0110001)\\ &\vdash_M (q_0, 110001)\\ &\vdash_M (q_1, 10001)\\ &\vdash_M (q_2, 0001)\\ &\vdash_M (q_2, 001)\\ &\vdash_M (q_2, 01)\\ &\vdash_M (q_2, 1)\\ &\vdash_M (q_1, \varepsilon)\end{aligned}$$

Since $(q_0, 00110001) \vdash_M^* (q_1, \varepsilon)$ and $q_1 \in F$, the string 00110001 is accepted by M. From the state diagram it is easy to conclude that strings of language $L(M)$ of automata M corresponds to the regular expression, $0^*1(00^*1 + 10^*1)^*$.

□

Often we reserve the length of a string in programming languages, but, this program, representing the transition diagram of a finite automaton for regular expression 0^*10^*, size of input may be of arbitrarily large. Hence, space for input cannot be allocated in advance. Fortunately, C language has standard *input function*, which can receive inputs of arbitrary length.

2.1.3 Extended Transition Function

Let an input string *abb* causes a FA to go through states q_1, q_2, and q_3 in order, from its initial state q_0. These transitions can be specified by,

1. $\delta(q_0, a) = q_1$
2. $\delta(q_1, b) = q_2$
3. $\delta(q_2, b) = q_3$.

Alternatively, q_1 in (2) can be substituted by its value from (1), which results to,

$$\delta(\delta(q_0, a), b) = q_2.$$

When value of this q_2 is substituted in (3), we get an expression:

$$\delta(\delta(\delta(q_0, a), b), b) = q_3.$$

This shows a case of multiple application of transition function δ, equal to length of the input string. We rewrite the above expression as,

$$\delta^*(q_0, abb) = q_3,$$

and call δ^* as *extended* transition function, and define it as,

$$\delta^* : Q \times \Sigma^* \mapsto Q. \tag{2.7}$$

Also, $\delta^*(q_0, w) \in F$, if w is accepted by the finite automaton.

Definition 2.2 *If all the strings w are accepted by a DFA M, then language of M is,*

$$L(M) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\} \tag{2.8}$$

and the complement of the language of M is,

$$\overline{L(M)} = \{w \in \Sigma^* \mid \delta^*(q_0, w) \notin F\} \tag{2.9}$$

which is set of all the strings in Σ^* but not in $L(M)$. In other words, the finite automata accepting complement of a language is new finite automata with final and non-final states interchanged.

Example 2.3 *Design a deterministic finite automaton M that accepts the language $L(M) = \{w \in a, b^* \mid w$ does not contain two consecutive bs$\}$.*

Let the finite automata which recognizes this language be $M = (\Sigma, Q, \delta, s, F)$. The string $w \in L(M)$ does not contain two consecutive bs, but it may contain any number of b and any number of a. Once the FA receives two consecutive bs at input, it should reach to a state from where it cannot reach to The final state.

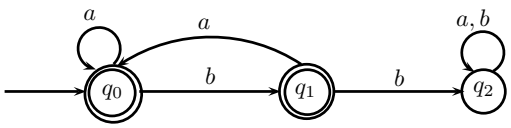

Figure 2.4: A DFA.

Figure 2.4 shows the DFA accepting this language. Various tuples of M are given below, and table 2.2 is transition table for this FA.

Table 2.2: Transition table for function δ.

Previous state q	Current input a	Current input b
q_0	q_0	q_1
q_1	q_0	q_2
q_2	q_2	q_2

All language strings of $L(M)$, for the above finite automaton can be described by a regular expression $(a + ba)^*(b + \phi)$, and,

$$L(M) = \{\phi, a, b, ab, ba, bab, \dots\}.$$

Example 2.4 *How many DFAs with two states can be constructed over an alphabet of* $\Sigma = \{a, b\}$*, with designated initial state.*

For each state, there can be $|\Sigma|$ number of edges coming out, so there are total $|\Sigma| \times |Q|$ edges possible ways edges can be connected o $|Q|$ states. So total DFAs are $2^{\Sigma Q} = 2^{2\times 2} = 2^4 = 16$. The designated states can be: state 1 start, 2 as final, the reverse of this case, both final, and non as final. So total DFAs are $4 \times 16 = 64$.