4CS4-6: Theory of Computation (Nondeterministic finite automata)

Lecture 04: Jan. 16 & 20, 2020

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

4.1 Introduction

Mathematics never saw much of a reason to deal with something called nondeterminism. It works with values, functions, sets, and relations. In computing science, however, nondeterminism has been an issue from the very beginning, if only in the form of *nondeterministic Turing machines* or *nondeterministic finite state machines.* Nondeterminism arises in a natural way when discussing concurrency, and models of concurrency, which typically also model nondeterminism. There are also numerous variants of process languages and algebra, event structures, state transition systems, which concerns to nondeterminism.

In terms of modeling, nondeterminism may be considered a purely operational notion. However, one of the main reasons for considering nondeterminism in computer science is the need for abstraction, allowing one to disregard irrelevant aspects of actual computations. Typically, we prefer to work with models that do not include all the details of the physical environment of computations such as timing, temperature, representation on hardware, and the like. Since we do not want to model all these complex dependencies, we may instead represent them by nondeterministic choices. The nondeterminism of concurrent systems usually arises as an abstraction of time. Similarly, nondeterminism is also a means of increasing the level of abstraction when describing sequential programs, and as a way of indicating a "don't care" attitude as to which among a number of computational paths will actually be utilized in a particular evaluation.

4.2 Model of Nondeterministic Finite Automata

All the computer programs as well as the model of Finite Automata (FA) introduced in the previous chapter are models of determinable computation. That is, in every configuration it is determined without any ambiguity as what will happen in the next computation step. Hence, a program or deterministic finite automata (DFA) can always determine unambiguously the computation over an input string x.

In contrast, a nondeterministic computation allows in each configuration a choice from finitely many actions. This has the consequence that a nondeterministic program can have many different computations on the same input. The only requirement is that one of them should ultimately yield correct result. This corresponds to an artificial rule: a nondeterminictic program always chooses a correct computation. A choice from several possibilities is called nondeterministic decision. The acceptability criteria of NFA (Non-deterministic Finite Automata) is as follows: with an alphabet set Σ, a string $w \in \Sigma^*$ leads to a sequence of configurations, where one of the configuration is an accepting configuration.

Although, an NFA being model of a nondeterministic program does not seem to be useful for practical purposes, but the study of nondeterminism will certainly contribute to our understanding of the nondeterministic computations.

Like DFA, an NFA is also a five tuple represented by,

$$M = (Q, \Sigma, \delta, s, F) \tag{4.1}$$

where,

Q is finite set of states,

Σ is finite set of alphabets,

δ is set of transitions,

s is start state, and

F is set of final or accepting states.

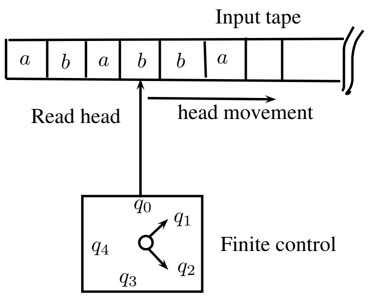

Figure 4.1: Nondeterminictic finite automata.

Since next state after a transition in an NFA is a state-set instead of a single state, and there can be 2^Q state sets for Q states, a state in an NFA is member of 2^Q states. Its transition function is defined as,

$$\delta : Q \times \Sigma \mapsto 2^Q. \tag{4.2}$$

Due to the same reason the final states are also members of 2^Q or $F \in 2^Q$, and $F \subseteq Q$. An NFA (Figure 4.1) consists infinite tape holding a string $w \in \Sigma^*$, and a read head, which can move to one direction only and reads the symbols on tape, square by square. The finite control of NFA in the Figure 4.1 indicates that in a particular configuration the machine has taken a move into two states together, i.e., q_1 and q_2, accordingly the next state is a set shown as $\{q_1, q_2\}$.

In an NFA the next state in a move is always a set, of size greater or equal to one. Hence, DFA is a special case of NFA, and the NFA is a generalized case of DFA. Thus, in NFA we may have,

$$\delta(q, a) = \{q', q'', q'''\} \subseteq Q. \tag{4.3}$$

Example 4.1 *Figure 4.2 shows the transition diagram of an NFA. When an input symbol $a \in \Sigma$ read at state q_0, the next state is $\delta(q_0, a) = \{q_1, q_2, q_0\}$, where all the three states are equally likely. This is because after reaching to state q_1 the NFA can move to state q_2 without an input, and from there it can come to state q_0 again by ε-transition.*

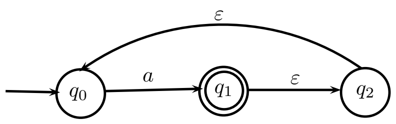

Figure 4.2: Transition diagram for an NFA.

An NFA accepts a string w if it leads to a sequence of moves with at least one sequence has the end element a final state. The string is rejected if there are no possible sequences of moves that can lead to a final state.

The equation (4.3) indicates that next state is a set of states in which there can be maximum $|Q|$ states at one time. For example, if there are three states $Q = \{q_0, q_1, q_2\}$ in an NFA, then the total number of next states can be 8 and a subset can have maximum three states. The possible sets of states in this case can be: $\{\}$, $\{q_0\}$, $\{q_1\}$, $\{q_2\}$, $\{q_1, q_2\}$, $\{q_2, q_3\}$, $\{q_1, q_3\}$, $\{q_1, q_2, q_3\}$.

A distinguishing feature of an NFA is - it can cause a transition without any input, that is, a null (ε) input can also cause a transition. Hence, the input symbol in NFA is member of $\Sigma \cup \{\varepsilon\}$. The general transition function δ for the NFA can therefore be defined as,

$$\delta : Q \times \Sigma \cup \{\varepsilon\} \mapsto 2^Q \tag{4.4}$$

The extended transition function δ^* for an NFA can be defined in the same lines of DFA,

$$\delta^*(q_i, w) = Q_j, \tag{4.5}$$

where $Q_j \subseteq Q$, $w \in \Sigma^*$, and Q_j is set of all the possible states in which NFA may exist when it had started at state q_i and have completely read the string w. Thus, the language accepted by an NFA is,

$$L(M) = \{w \in \Sigma^* \mid (q_0, w) \vdash^* (Q', \varepsilon)\} \tag{4.6}$$

where $Q' = \{..., q_f, ...\} \in 2^Q$, $q_f \in F$.

The non-deterministic automata are not model of real computers but is simply useful notational generalizations of finite automata. They are useful because they greatly simplify the description of finite automata. We will show later that an NFA is equivalent to DFA and at the same time it appears in much simplified form. When an NFA has more than one possible next state, it can be viewed as more than one DFA working in parallel.

4.3 Properties of an NFA

We will see that, there are some qualities of NFA, and that is why at the initial state a system is constructed as an NFA, and then later is is converted into DFA. Subsequently, the DFA is simplified by minimization.

4.3.1 NFA is Simpler

Designing an NFA is simpler than DFA, because an NFA is far less complex than a DFA. Consider the language

$$L = \{w \mid w \in (ab + aba)^*\}$$

with alphabet set $\Sigma = \{a, b\}$. The DFA that recognizes language L is shown in figure 4.3. We note that, the regular expression and the corresponding language appear to be simple. However, the corresponding deterministic finite automaton is complex. It can be shown that it is not possible to obtain a DFA of less than five states for this regular expression.

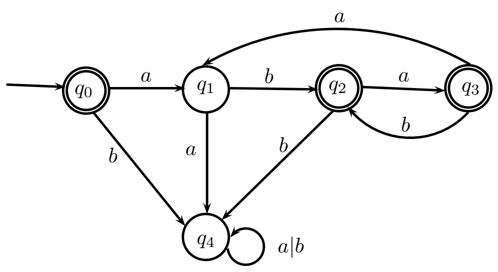

Figure 4.3: DFA for language $(ab + aba)^*$.

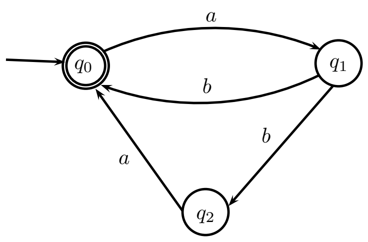

Figure 4.4: NFA-I for language $(ab + aba)^*$.

The language strings generated by the regular expression $(ab + aba)^*$ are also recognized by an NFA shown in figure 4.4. However, there also exist paths in NFA using which the same string cannot be recognized. For example, for string aba, the path q_0, q_1, q_2, q_0 recognizes it. But q_0, q_1, q_0, q_1 does not. The requirement for recognition of a string by NFA is that there should be some path(s) available for a walk-through that ultimately lead to a final state when complete string is read.

For an NFA, unlike the DFA, it is not necessary that all transitions corresponding to inputs are available at each state. For example in figure 4.4, at state q_0 and q_2 input b is not accepted. Hence $\delta^*(q_0, abb) = \phi$, as there is no state available after receiving the string *abb*. Such configurations are called *dead configurations*. We note that, in spite of dead configurations, the NFA in figure 4.4 is a much simpler than the DFA shown in figure 4.3.

Equally simpler NFA is shown in figure 4.5, which can also recognize the language of regular expression $(ab + aba)^*$. This NFA includes a null transition at state q_2, i.e., transition occurring in the absence of any input.

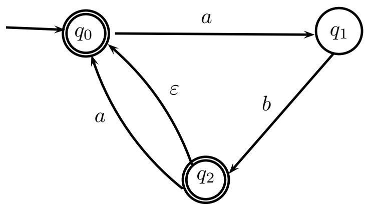

Figure 4.5: NFA-II for language $(ab + aba)^*$.

4.3.2 An NFA simulates more than one DFA in Parallel

Computers are deterministic machines and their final states are predictable if their initial state and input are known. However, for an NFA as computing model, every next state is a set (often of more than one) of states, thus NFA is not predictable. This drawback of NFA is blessing as it helps in faster problem solving by carrying many states together.

Any problem solution requires the exploration of number of states. All these states put together is called *state space*. To solve a problem, we exhaustively search this state space to find out if it leads to final state. In the process of searching, we may sometime reach to a state from where solution is not possible. In that case, we backtrack to some previous state and again search in the state space. This *backtrack* might repeat number of times depending on the size of state space and nature of the problem being solved [floyd97].

In an NFA, while searching the state space, we carry on number of states together, and one or more of them may lead to the final state, and hence to the solution. Since, backtracking is not required in an NFA, it may lead to a faster and efficient solution.

Example 4.2 *For the NFA shown in figure 4.6, find out whether the string $w = aab$ is accepted by this NFA.*

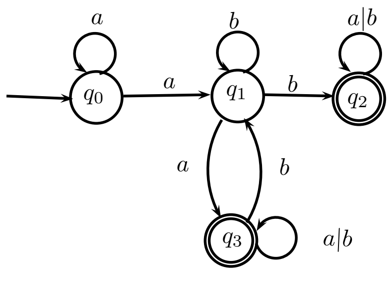

Figure 4.6: NFA.

We show the sequence of configurations through which this NFA moves when it reads the input.

$$\begin{aligned}
(q_0, aab) &\vdash (\{q_0, q_1\}, ab) \\
&\vdash (q_0, ab) \cup (q_1, ab) \\
&\vdash (\{q_0, q_1\}, b) \cup (q_3, b) \\
&\vdash (q_0, b) \cup (q_1, b) \cup (q_3, b) \\
&\vdash (\phi, \varepsilon) \cup (\{q_1, q_2\}, \varepsilon) \cup \{\{q_1, q_3\}, \varepsilon\} \\
&\vdash (q_1, q_2, q_3), \varepsilon)
\end{aligned}$$

Figure 4.7, which shows the transitions in NFA of figure 4.6, which gives the impression of more than one DFA operating in parallel.

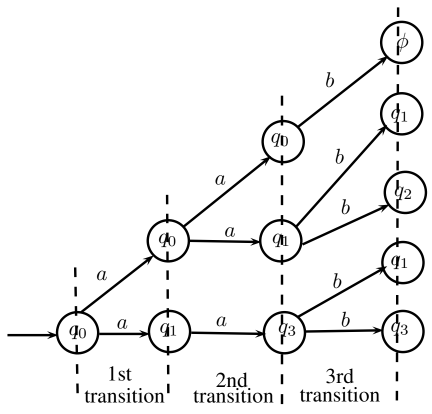

Figure 4.7: Transitions of NFA of figure 4.6 for input *aab*.