lec3:Nondeterministic finite state automata

1.introduction

Nondeterminism is a useful concept that has great impact on the theory of computation. When the machine is in a given state and reads the next input symbol, we know what the next state will be -it is determined this called deterministic computation but in a nondeterministic machine, several choices may exit for the next state at any point.

Nondeterminism is a generalization of determinism ,so every deterministic finite automaton is automatically a nondeterministic finite automaton. The different between a deterministic finite automation and a nondeterministic finite automaton.

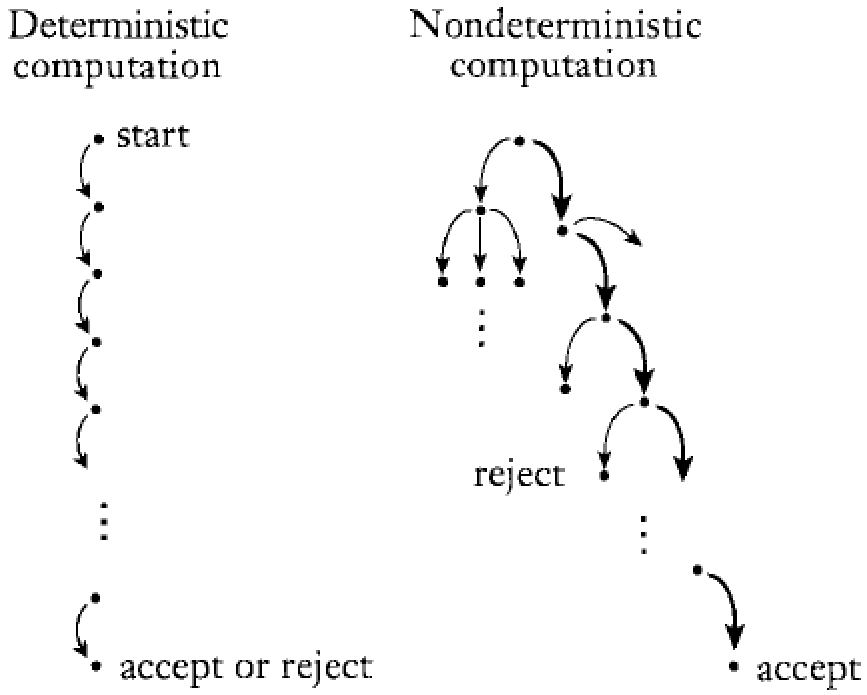
1- every state of always has exactly one exiting transition arrow for each symbol in the alphabet but the nondeterministic violates that rule.

2-In DFA, label on the transition arrow are symbols from the alphabet .this NFA has an arrow with the label ϵ , an NFA may have arrows labeled with members of alphabet or ϵ. Zero, one , or many arrows may exit from each state with the label ϵ

we are running an NFA on an input string and come to a state with multiple ways to proceed, the machine splits into multiple copies of itself and follows all the possibilities in parallel. Each copy of the machine takes one of the possible ways to proceed and continues as before. if there are subsequent choices, the machine splits again. if the next input symbol dose not appear on any of the arrow exiting the state occupied by a copy of the machine ,that copy of the machine dies, along with the branch of the computation associated with it. if any one of these copies of the machine is an accept state at the end of the input, the NFA accepts the input string. if state with an ϵ symbol on an exiting arrow is encountered, something similar happens. without reading any input, the machine splits into multiple copies, one following each of the exiting ϵ-label arrows and one staying at the current state.

Nondeterminism may be viewed as a kind of parallel computation wherein multiple independent "processes" or "thread" can be running concurrently. when the NFA splits to following several choices, that corresponds to a process "forking" into sveral children, each proceeding separately. if at least one of these processes accepts, then te entire computation accepts.

Another way to think of a nondeterministic computation is as tree of possibilities. the root of the tree corresponds to the start of the computation. Every branching point in the tree corresponds to a point in the computation at which the machine has multiple choices. the machine accepts if at least one of the computation branches ends in an accept state. as show below.

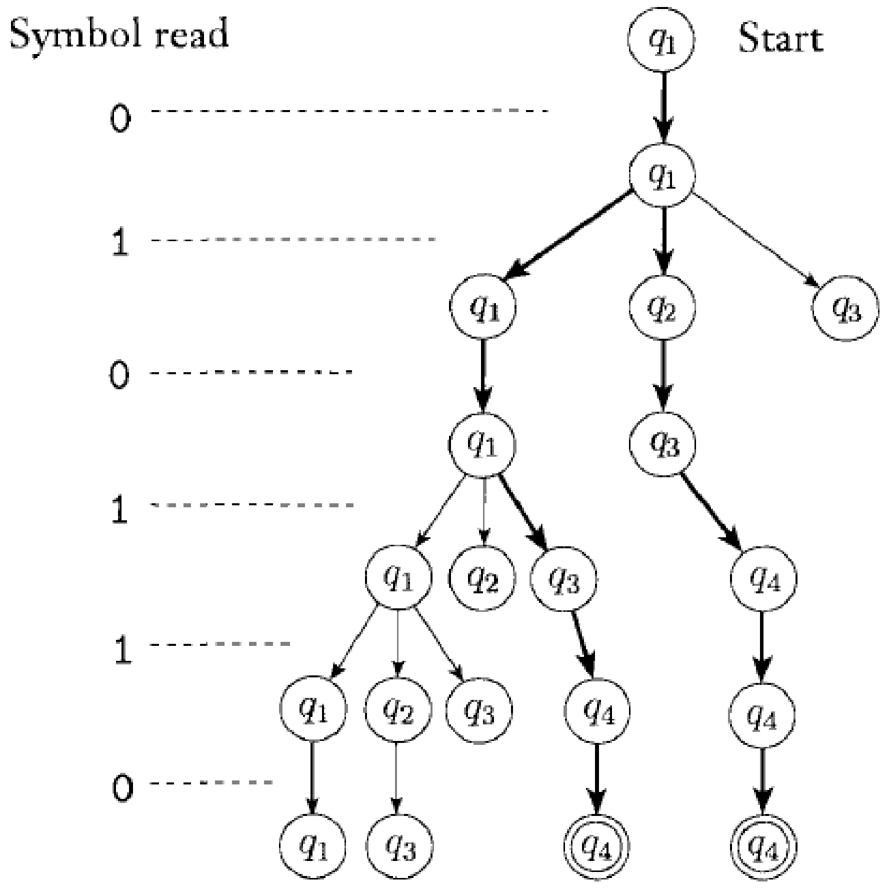


Example: state q1 has one exiting arrow for 0, but it has two for 1; state q2 has one arrow for 0, but it has none for 1. in an NDF a state may have zero, one, or many exiting arrows for each alphabet symbol .as show in figure below machine N1

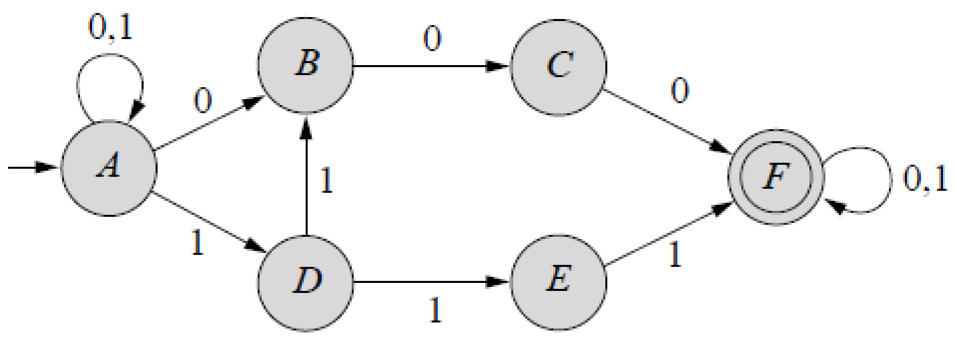
$$\begin{tikzcd}
\arrow[r] & q_1 \arrow[loop above, "0,1"] \arrow[r, "1"] & q_2 \arrow[r, "0{,}\varepsilon"] & q_3 \arrow[r, "1"] & q_4 \arrow[loop above, "0,1"]
\end{tikzcd}$$

.

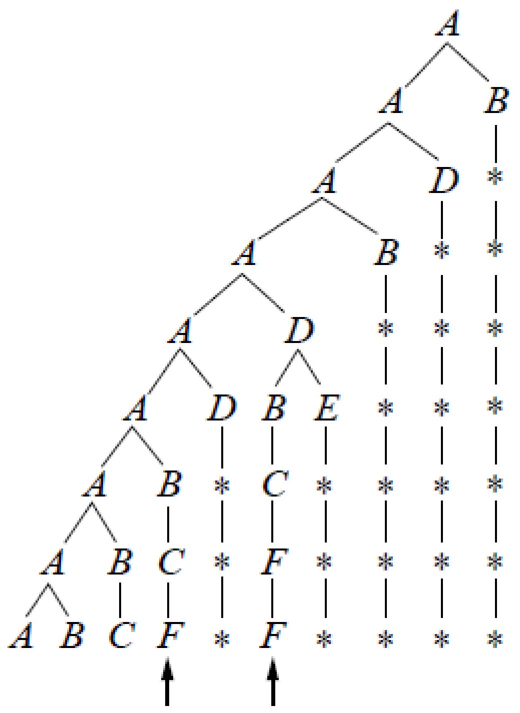
the computation of N1 on input 010110 is depicted in the following



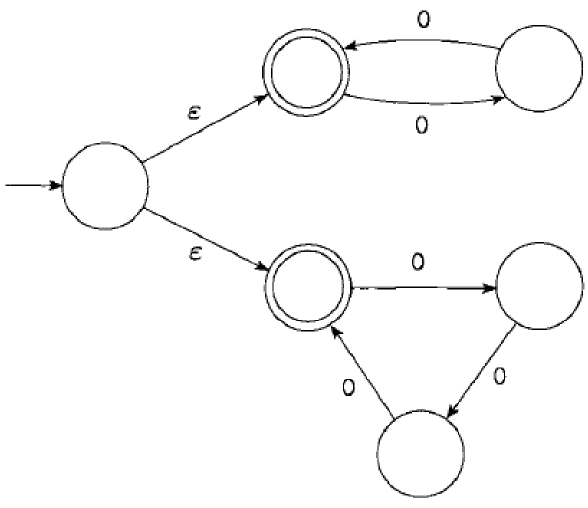
Example: Consider the nondeterministic finite automaton of Figure below, which accepts all strings that contain one of three possible substrings: 000, 111, or 1100.



The computation tree on the input string 01011000 is de-picted in Figure below. (The paths arked with an asterisk denote paths where the automaton is stuck in a state because it had no transition available.) There are two accepting paths out of ten, corresponding to the detection of the substrings 000 and 1100. The nondeterministic finite automaton thus accepts 01011000 because there is at least one way (here two) for it to do so.

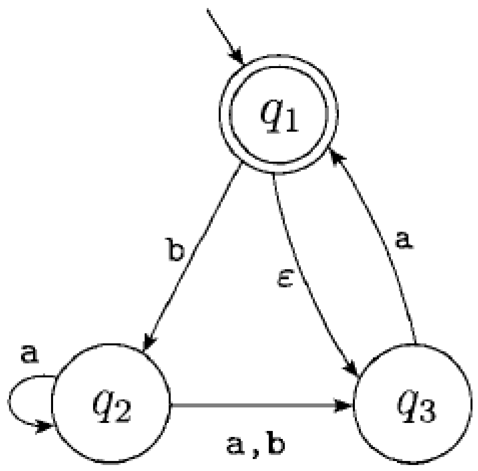


Example : consider the following NFA below that has an input alphabet {0} consisting of a single symbol. An alphabet containing only symbol is called a unary alphabet.



this machine demonstrates the convenience of having ϵ arrows. It accepts all strings of the from 0^k where k is a multiple of 2 or 3. for example machine accepts the strings ϵ, 00, 000, 0000, and 000000, but not 0 or 00000.

Example: we given of NFA in following figure below, practice with it to satisfy yourself that it accept the string ϵ, a, baba, and baa, but that it doesn't accept the string b, bb, and babba.



2. Formal Definition of a NFA

the formal definition of a nondeterministic finite automaton is similar to that of a deterministic finite automaton. Both have states, an input alphabet, transition function, start state, and a collection of accept states..

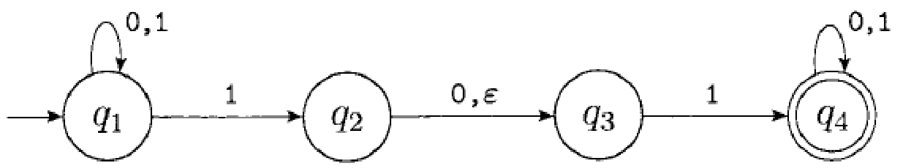
A ***nondeterministic finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta \colon Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

Example :

....

Recall the NFA N_1:



The formal description of N_1 is $(Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3, q_4\}$,
2. $\Sigma = \{0,1\}$,
3. δ is given as

	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

,

4. q_1 is the start state, and
5. $F = \{q_4\}$.

the formal definition of computation for an NFA is similar to that for a DFA. Let N=(Q, ∑, δ,q_0, F) be an NFA and ω a string over the alphabet ∑. then w say that accept ω if we can write ω as

ω=y1y2y.......y_m where each y_i is a member of ∑ε and a sequence of state $r_{0,}$ $r_{1,}$ ....,$r_{m.}$

exists in Q with three conditions:

1. $r_0 = q_0$,
2. $r_{i+1} \in \delta(r_i, y_{i+1})$, for $i = 0, \ldots, m-1$, and
3. $r_m \in F$.

condition 1 says that the machine starts out in the start state. condition 2 says that state r_{i+1} is one of the allowable next states when N is in state r_i and reading y_{i+1} . Observe that δ($r_{i,}$ $y_{i+1,}$) is the set of allowable next states and so we say that r_{i+1} is a member of that set.

3. Equivalence of NFA to DFA

Deterministic and nondeterministic finite automata recognize the same class of languages. Such equivalence is both surprising and useful. it is surprising because NFA appear to have more power than DAF, so we might expect that NFA recognize more language. Say that two machine are equivalent if they recognize the same language. every nondeterministic finite automaton has an equivalent deterministic finite automaton.

the definition nondeterministic finite automata include deterministic ones as a special case the case where the number of transitions defined for each pair of current state and current input symbol never exceeds one. Thus any language that can be accepted by a deterministic finite automaton can be accepted by a nondeterministic one the same machine.

we construct a deterministic one that mimics the behavior of the nondeterministic machine. In particular, the deterministic machine uses its state to keep track of all of the possible states in which the nondeterministic machine could find itself after reading the same string.

Theorem : For every nondeterministic finite automaton, there exists an equivalent deterministic finite automaton (i.e., one that accepts the same language).

Proof. Let the nondeterministic finite automaton be given by the five- tuple (Q, ∑, δ,q_0, F). We construct an equivalent deterministic automaton (Q', ∑', δ',q_0', F') as follows:

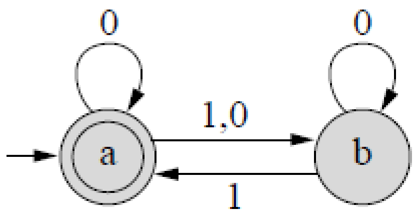
$$\begin{aligned}
&\Sigma' = \Sigma \\
&Q' = 2^Q \\
&F' = \{s \in Q' \mid s \cap F \neq \emptyset\} \\
&q_0' = \{q_0\}
\end{aligned}$$

Example: Consider the nondeterministic finite automaton given by

$$\Sigma = \{0,\ 1\},\ Q = \{a,\ b\},\ F = \{a\},\ q_0 = a,$$

$$\delta: \quad \begin{array}{ll} \delta(a,\ 0) = \{a,\ b\} & \delta(a,\ 1) = \{b\} \\ \delta(b,\ 0) = \{b\} & \delta(b,\ 1) = \{a\} \end{array}$$

and illustrated in Figure below (a). The corresponding deterministic finite automaton is given by

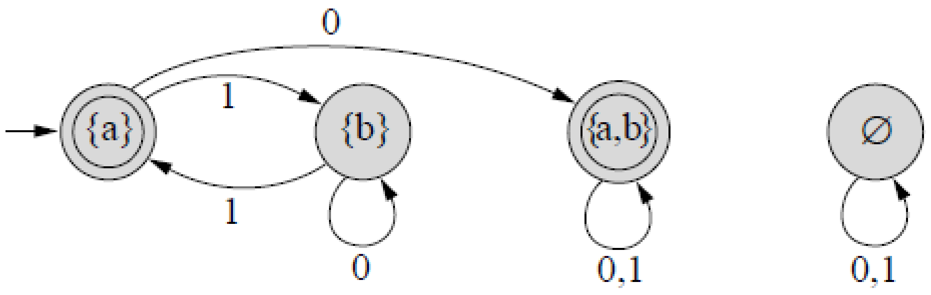


(a) the nondeterministic finite automaton

$$\Sigma = \{0, 1\},\ Q' = \{\emptyset, \{a\}, \{b\}, \{a, b\}\},\ F' = \{\{a\}, \{a, b\}\},\ q'_0 = \{a\},$$

$$\delta' : \begin{array}{ll} \delta'(\emptyset, 0) = \emptyset & \delta'(\emptyset, 1) = \emptyset \\ \delta'(\{a\}, 0) = \{a, b\} & \delta'(\{a\}, 1) = \{b\} \\ \delta'(\{b\}, 0) = \{b\} & \delta'(\{b\}, 1) = \{a\} \\ \delta'(\{a, b\}, 0) = \{a, b\} & \delta'(\{a, b\}, 1) = \{a, b\} \end{array}$$

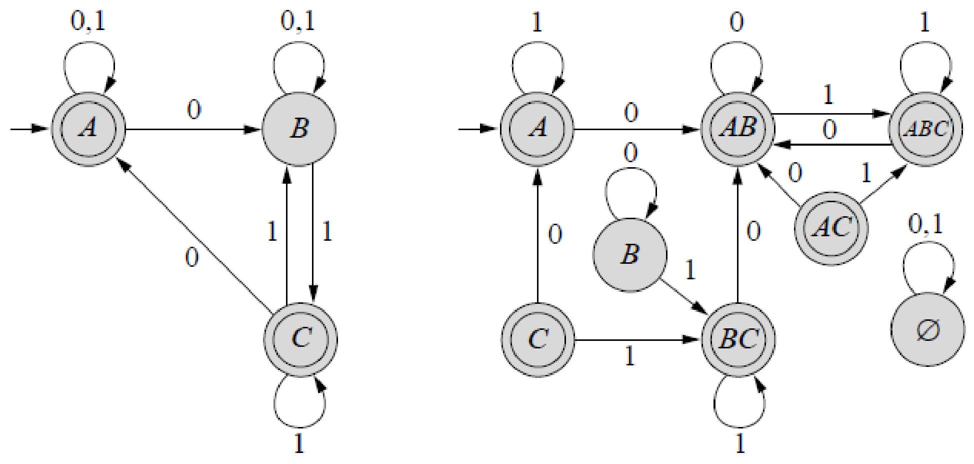
and illustrated in Figure below (b) (note that state ∅ is unreachable).



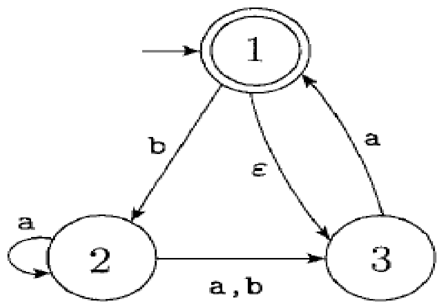
(b) the equivalent deterministic finite automaton

Thus the conversion of a nondeterministic automaton to a deterministic one creates a machine, the states of which are all the subsets of the set of states of the nondeterministic automaton. The conversion takes a nondeterministic automaton with n states and creates a deterministic automaton with 2n states, an exponential increase. we saw briefly, many of these states may be useless, because they are unreachable from the start state; in particular, the empty state is always unreachable. In general, the conversion may create any number of unreachable states,

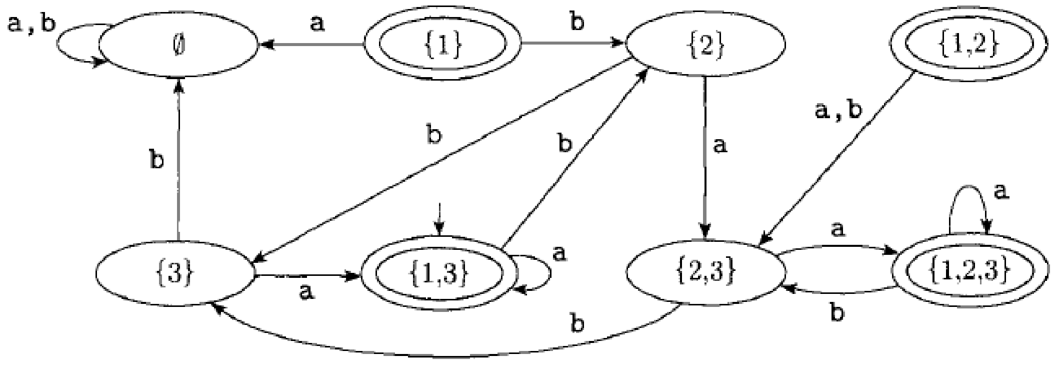
Example: convert NFA to DFA for machine below



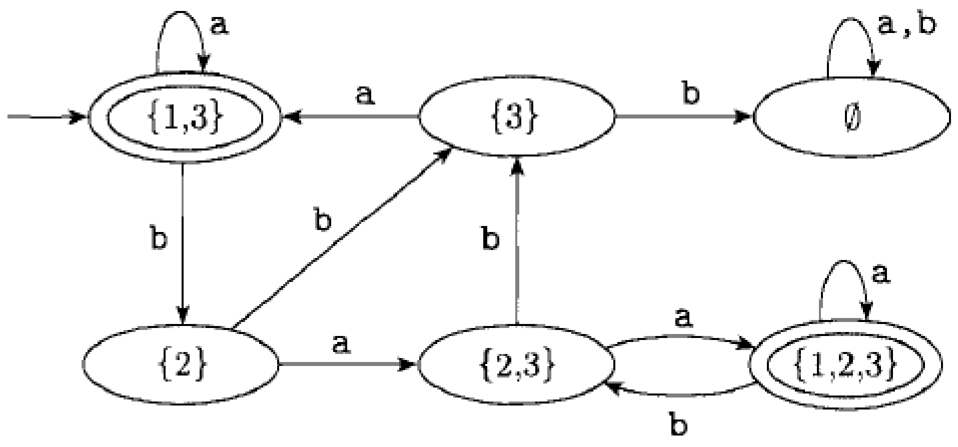
Example: converting NFA to DFA for machine below:



a:NFA



b:DFA



c:DFA after remove unnecessary state

When generating a deterministic automaton from a given nondeterministic one, we can avoid generating unreachable states by using an iterative approach based on reachability : begin with the initial state of the nondeterministic automaton and proceed outward to those states reachable by the nondeterministic automaton. This process will generate only useful states—states reachable from the start state—and so may be considerably more efficient than the brute-force generation of all subsets.