§1. Deterministic Finite State Acceptors

A Finite State Machine (FSM) is an abstract model of a digital computing machine in which the device can enter into only a finite number of different states. In a finite state machine, there may be input, storage, or output tapes of finite or unlimited capacities. Information can be read from or written on these tapes by a control unit (which is also known as the head).

Finite state machines can be classified according to the kinds of tape they have, the kind of access to any storage tape they may have, and the mode of operation.

Def. An acceptor is an FSM with an input tape and a yes/no output tape. (There may or may not be any storage tape.) A yes/no output tape has only one space for a "0" (NO) or a "1" (YES).
## Classification of Finite State Machines

<table>
<thead>
<tr>
<th>TYPE</th>
<th>TAPES, STORAGE PRESENT, MODE OF OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFA</td>
<td>Infinite input tape, no storage tape, Yes/No output tape, Deterministic mode of operation</td>
</tr>
<tr>
<td>NFA</td>
<td>Infinite input tape, no storage tape, Yes/No output tape, Non-deterministic mode allowed.</td>
</tr>
<tr>
<td>DPDA</td>
<td>Infinite input tape, infinite storage tape with stack access only, Yes/No output tape, Non-deterministic mode of operation</td>
</tr>
<tr>
<td>PDA</td>
<td>Infinite input tape, infinite storage tape with stack access only, Yes/No output tape, Non-deterministic mode allowed</td>
</tr>
<tr>
<td>LBM</td>
<td>Infinite storage tape which also functions as the input &amp; output tape, random access to a portion of the storage tape which is bounded by a constant times the size of input, Non-deterministic mode allowed.</td>
</tr>
<tr>
<td>DTM</td>
<td>Infinite storage tape which functions as the input &amp; output tape, random access to the whole of the storage tape, Deterministic mode</td>
</tr>
</tbody>
</table>

Same as DTM except Non-deterministic mode allowed.
Ex. 1. Suppose we want an FSM which does binary addition. How should we design it?

\[ \begin{array}{c}
0101001 \\
011011 \\
\hline
1000100
\end{array} \quad \begin{array}{c}
\text{BINARY} \\
\text{ADDER} \\
\rightarrow \\
\rightarrow 
\end{array} \\
\begin{array}{c}
(0)/0 \\
(1)/0 \\
(0)/1 \\
(1)/1
\end{array} \\
\begin{array}{c}
(0)/0 \\
(1)/0 \\
(0)/1 \\
(1)/1
\end{array}

\text{Input: } (0)(1)(0)(1)(0)(0)(1)(1) \quad \text{Output: } 1000100

\text{State: NCCCCNCCC} \quad \text{starting state}

\underline{Note:} The FSM we got is not an acceptor but its mode of operation is deterministic. So we can call it a DFSM.

\text{Def. A Deterministic Finite Acceptor (DFA) is a 5-tuple } M = \langle Q, T, \delta, q_0, A \rangle \text{ where}

\begin{align*}
&Q(M) = Q \text{ is a finite set of (internal) states} \\
&T(M) = T \text{ is an alphabet called the input alphabet} \\
&\delta_M = \delta : Q \times T \rightarrow Q \text{ is a function called the transition function} \\
&q_0(M) = q_0 \in Q \text{ is the designated initial state, and} \\
&A(M) = A \subseteq Q \text{ is the designated set of accepting states.}
\end{align*}

Ex. 1. Let \( M_1 = \langle Q, T, S, q_0, A \rangle \) where \( Q = \{ A, B, C \} \), \( T = \{ 0, 1 \} \), \( q_0 = A \), \( A = \{ B \} \) and \( S : Q \times T \rightarrow Q \) is defined by
\[ \begin{align*}
S(A, 0) &= A & S(A, 1) &= B \\
S(B, 0) &= A & S(B, 1) &= C \\
S(C, 0) &= C & S(C, 1) &= B
\end{align*} \]

We can represent \( M \) by the transition table on the right with "\( \rightarrow \)" designating the initial state & "\( \bigcirc \)" designating accepting states.

We can also represent \( M \) by a transition graph with "\( \rightarrow \)" designating the initial state & "\( \bigcirc \)" designating the accepting states.

Note:

It is possible for a state to be the initial state as well as an accepting state. If \( S(B, 0) = D \) in \( M \)
we say that \( B \rightarrow D \) is a transition in \( M \).

We can describe the operation of a DFA \( M \) by giving the sequence of configurations of \( M \) when it is presented with an input \( \varphi \in \Sigma^{*} \). A configuration is an ordered pair \( \langle q, \varphi_{1} \varphi_{2} \rangle \) where \( q \) is the control state and \( \varphi_{1} \varphi_{2} \) is the input string with the head under the beginning of \( \varphi_{2} \).

\[ \varphi_{1}, \varphi_{2} \]

\text{tape:} \[ a \ b \ a \ c \ b \ b \ a \ w \ldots \]

\text{head in state} \( B \) \text{ in position indicate configuration:} \[ \langle B, \text{abacbbb} \rangle \]

\( \varphi_{1} \equiv \varphi_{2} \)

\( \text{blank symbol} \)
Let us run the DFA \( M \), of Ex. 1 with \( q=1101 \) as input. When the head reaches a "\( \omega \)" \( M \) will halt.

\[
\begin{align*}
\langle A, 1101 \rangle & \rightarrow \langle B, 1 \rangle \\
& \rightarrow \langle C, 1 \rangle \\
& \rightarrow \langle C, 1 \rangle \\
& \rightarrow \langle B, 1 \rangle
\end{align*}
\]

The tapes will look as shown below:

\[
\begin{array}{c}
\text{A} \\
\hdashline
\text{B} \\
\hdashline
\text{C} \\
\hdashline
\text{C}
\end{array}
\]

Notice that the input does not change - we simply move into a control state with each character of the input and stop (halt) as soon as we reach the blank symbol "\( \omega \)."

If the last control state we reach is an accepting state, we say that the DFA accepts the input; if it is not an accepting state, we say that the DFA rejects the input.

So 1101 is accepted by the DFA \( M \) in Ex. 1.
Note: In a DFA we are only allowed to input a single string $q$ from $T^*$. The "|" cannot be a part of the input alphabet $T$. Instead of showing the sequence of configurations of the DFA $M_i$ on an input $q$ (or the sequence of tapes) we can just show what happens using two rows as shown below.

| input characters: | 1 | 1 | 0 | 1 | | |
|------------------|---|---|---|---|---|
| control states:  | A | B | C | C | B |

Let us try another input, say $q = 100$ on $M_1$.

| input: | 1 | 0 | 0 | | |
|--------|---|---|---|---|
| states: | A | B | A | A |

Since $A$ is not an accepting state, $100$ is not accepted by $M$. We will define $L(M)$ to be the set of all strings in $T^*$ which are accepted by $M_i$.

**Ex 2** Let $M_2$ be the DFA shown on the right. $M_2$: 

Find $L(M_2)$.

$L(M_2) = 0^*11^*$

**Ex 3** Let $M_3$ be the DFA shown below. Find $L(M_3)$.

$L(M_3) = (1 + 01)^*$
We will now define precisely what it means for a DFA $M$ to accept a string $q$, and precisely what is $L(M)$.

**Def.** The extended transition function $S^*: Q \times T^* \rightarrow Q$ of a DFA is defined recursively as follows.

(a) $S^*(q, \lambda) = q$
(b) $S^*(q, qc) = S(S^*(q, c), c)$ for each $c \in T$.

**Def.** The string $q \in T^*$ is said to be **accepted** by a DFSA $M$ if $S^*(q_0, q) \in A(M)$. The string $q$ is **rejected** by $M$ if $S^*(q_0, q) \notin A(M)$.

**Def.** The language recognized by a DFA $M$ is defined by $L(M) = \{ q \in T^* : S^*(q_0, q) \in A(M) \}$.

**Ex. 1** Find a DFA $M_1$ which recognizes the language $L_1 = \{ a^n b : n \geq 0 \}$.

**Ex. 2** Find a DFSA $M_2$ which recognizes the language $L_2 = a b^* b a^*$.  
Hint: $a b^* b a^* = a b b^* a^*$.
Ex. 3
Let \( n_a(\omega) \) = number of a's in \( \omega \) and \( n_b(\omega) \) = the number of b's in \( \omega \). Find a DFA which recognizes the language
\[ L_3 = \{ \omega \in \{a, b\}^* : n_a(\omega) + 2n_b(\omega) \equiv 3 \pmod{4} \} \]

Let the states be \( A_0, A_1, A_2, \) and \( A_3 \) and let \( f(\omega) = n_a(\omega) + 2n_b(\omega) \). We will use \( A_i \) to keep track of the fact that when the initial part \( \varphi \) of \( \omega \) is processed, the value of \( f(\varphi) = i \pmod{4} \). Since \( f(\lambda) = 0 \), the initial state will be \( A_0 \). Also when \( f(\varphi) = 3 \pmod{4} \), \( \varphi \) will be accepted, so \( A_3 \) is the only accepting state. Now
\[ f(p_a) = n_a(p_a) + 2n_b(p_a) = n_a(p) + 2n_b(p) + 1 = f(p) + 1 \pmod{4} \]
\[ f(p_b) = n_a(p_b) + 2n_b(p_b) = n_a(p) + 2n_b(p) + 2 = f(p) + 2 \pmod{4} \]

So \( M_3 = \)

Ex. 4
Show what happens when \( \omega = ababa \) is used as the input to the DFA \( M_3 \) and verify your answer using modulo arithmetic.

Input: \( a \ b \ a \ b \ a \)

States: \( A_0, A_1, A_3, A_0, A_2, A_3 \)

So we \( L(M_3) \)

Check: \( f(\omega) = n_a(\omega) + 2n_b(\omega) = 3 + 2(2) = 7 \equiv 3 \pmod{4} \)
Since we \( L_3 \), \( \omega \) should be accepted by \( M_3 \).
§2. The Partition Algorithm & Minimal DFSA's

Def. Two DFSA's $M_1$ and $M_2$ are said to be equivalent if $L(M_1) = L(M_2)$.

Ex.1 Consider the two DFSA's $M_1$ & $M_2$ below

It is not difficult to see that $L(M_1) = (0+1)^* 1 (0+1)^* = L(M_2)$. So $M_1 \equiv M_2$.

But it is clear that $M_2$ is much simpler than $M_1$. We will later see $M_2$ is the DFSA with the smallest number of states which is equivalent to $M_1$.

Def. A state $q$ in a DFA $M$ is said to be inaccessible if there is no string $w \in T^*$ such that $s^*(q_0, w) = q$.

In other words, $q$ is inaccessible if we can never reach it from the initial state. In $M_1$, the state $F$ is inaccessible because there is no way to get from $A$ to $F$.

Ex.2 How can we find all the inaccessible states in the DFSA $M_1$ of Ex.1?
Start with the initial state of \( M \) and keep branching out with each character of \( T \). When a previously accessible state is reached, discontinue that branch. The accessible states of \( M \) will be all the states in the accessibility tree.

\[
\begin{array}{c}
& & & & D_x \\
& & O & \rightarrow & C_x \\
& B & \rightarrow & C & \leftarrow & E \\
A & \rightarrow & D_x & \leftarrow & C & \leftarrow & E \\
& D & \rightarrow & D_x & \leftarrow & E \\
& I & \rightarrow & D & \leftarrow & E \\
& I & \rightarrow & A & \leftarrow & D \\
\end{array}
\]

So the acc. states are \( A, B, C, D, \& E \).... \( F \) is inacc.

To find the DFA \( M_r \) which is equivalent to a given DFA \( M \), we will remove the inaccessible states and get rid of the redundant states.

\[\text{Def. Two states } p \& q \text{ in a DFA, } M, \text{ are indistinguishable if for each string } y \in T^* \text{ we have } S^*(p,y) \in A(M) \iff S^*(q,y) \in A(M)\]

So indistinguishable states are in a sense equivalent.

Also if \( p_1, p_2, \ldots, p_k \) are indistinguishable states which are all accessible in \( M \), then we need only one of them, \( p \), say, and the rest will all be redundant.

In \( M_r \), the states \( B \& D \) are indistinguishable. Also \( C \& E \) are indistinguishable. So if we keep \( B \) and \( C \), then \( D \) and \( E \) will be redundant & can be removed.
In removing a redundant state, we have to make sure that we replace it by an equivalent one.

The Partition Algorithm for DFA

Input: The set of states of a DFA $M$
Output: Blocks of indistinguishable states of $M$

1. Start with the initial partition $P_0$ consisting of 2 blocks $P_0: Q - A, A$. Then let $n \leftarrow 0$.

2. Two states $p$ and $q$ in the same block of $P_n$ will stay together in the same block of $P_{n+1}$ if for each $c \in T$, $s(p, c)$ & $s(q, c)$ belong to the same block of $P_n$. If for at least one $c \in T$, $s(p, c)$ & $s(q, c)$ belong to different blocks of $P_n$, then $p$ & $q$ will split and go into different blocks of $P_{n+1}$.

3. If $P_{n+1} = P_n$, then STOP. ($P_n$ will be the final partition of $Q$ into blocks of indistinguishable states. Otherwise, let $n \leftarrow n+1$; and go to step 2.

Ex. 3: Partition the states of the DFA $M_3$ into blocks of indistinguishable states.

$$M_3:$$

![Diagram of DFA $M_3$]

Notice that all the states in $M_3$ are accessible.
1. \( P_0 : \{ A, B, C, D \} \setminus \{ E \} \)
   - \( A \vdash \text{1st block of } P_0 \)
   - \( B \vdash \text{1st block of } P_0 \)
   - \( C \vdash \text{1st block of } P_0 \)
   - \( D \vdash \text{1st block of } P_0 \)
   - \( A \vdash \text{1st block of } P_0 \)
   - \( B \vdash \text{2nd block of } P_0 \)
   - \( C \vdash \text{2nd block of } P_0 \)
   - \( D \vdash \text{2nd block of } P_0 \)

2. \( P_1 : \{ A \} \setminus \{ B, C, D \} \setminus \{ E \} \)
   - \( B \vdash \text{2nd block of } P_0 \)
   - \( C \vdash \text{2nd block of } P_0 \)
   - \( E \vdash \text{3rd block of } P_0 \)
   - \( B \vdash \text{3rd block of } P_0 \)
   - \( C \vdash \text{3rd block of } P_0 \)
   - \( D \vdash \text{3rd block of } P_0 \)

3. \( P_2 : \{ A \} \setminus \{ B, C, D \} \setminus \{ E \} \)
   - Since \( P_2 = P_1 \), \( P_2 \) is the final partition of \( \Delta \) into blocks of indistinguishable states.

Ex. 4. Find the minimal (or reduced) DFA \( M_R \) that is equivalent to the DFA \( M_3 \) in Ex. 3.

We will use each block of the final partition as a state in \( M_R \). The initial state will be the one that contains the initial state of \( M_3 \). The accepting states will be the blocks which consist only of accepting states.

\[
M_R =
\begin{align*}
&\{A, B, C, D\} \xrightarrow{0, 1} \{B, C, D\} \\
&\{A, B, C, D\} \xrightarrow{1, 0} \{E\} \\
&\{A, B, C, D\} \xrightarrow{0, 1} \{E\}
\end{align*}
\]

Some textbooks like to pick one representative each to get

\[
\begin{align*}
&\{A\} \xrightarrow{0, 1} \{B\} \xrightarrow{1, 0} \{E\} \\
&\{A\} \xrightarrow{0, 1} \{B\} \xrightarrow{1, 0} \{E\}
\end{align*}
\]

but we prefer using the blocks.

Note: \( L(M_3) = L(M_R) = (0 + 1)^* \cdot 0^* \cdot (0 + 1)^* \).
Ex.5 Find the minimal DFA $M_k$ that is equivalent to the DFA $M$ below.

\[
\begin{array}{cccccccc}
 & A & B & C & D & E & F & G \\
0 & C & D & B & A & A & C & F \\
1 & F & B & F & G & E & A & D \\
\end{array}
\]

1. First we check for any inaccessible states

2. Now we partition $\{A, B, C, D, F, G\}$ into blocks of indistinguishable states.

- $P_0: \{A, B, F\}$  \quad $\{C, D, G\}$
  non-accepting states  accepting states

- $P_1: \{A, B, F\}$  \quad $\{C\}$  \quad $\{D, G\}$

- $P_2: \{A, F\}$  \quad $\{B\}$  \quad $\{C\}$  \quad $\{D, G\}$

- $P_3: \{A, F\}$  \quad $\{B\}$  \quad $\{C\}$  \quad $\{D, G\} = P_2$
  = final partition

3. Finally we make the DFA $M_k$.

\[
M_k = \begin{array}{c}
\{A, F\} \quad \{C\} \quad \{B\} \quad \{D, G\}
\end{array}
\]
§3. Non-deterministic Finite Acceptor

Ex. 1 Find a DFA which recognizes the language
\[ L_1 = (ab)^* + (ba)^* \]

Although \( L_1 \) appears to be a rather simpler language, the DFA \( M_1 \) is decidedly complex.

If we generalize the concept of a DFA and allow non-deterministic mode of operation, we can get a much simpler acceptor \( M'_1 \) which recognizes \( L_1 \).

We can jump from A to B with the empty string \( \lambda \) and go around the loop from B to C to recognize \( (ab)^* \) — but we can't go back to A from B and
try to reach D. We can jump with $\lambda$ from A to D and go around the loop from D to E to recognize the second part $(ba)^*$, but again we cannot go back to A from D. So $L(M^{'}) = \emptyset$.

**Note:**

In the NFA $M'$, $A \xrightarrow{\lambda} B$ and $A \xrightarrow{\lambda} D$ are transitions which are called lambda-transitions. Also if we reach C and the next character is an "a" then we cannot move and we say that the NFA crashes. (The NFA $M$ will accept a string $w$ only if there is some way to reach the first blank symbol $\omega$ after $w$ and do so in a accepting state). Also for each state $x$, the transition $x \xrightarrow{\lambda} x$ is present but not shown.

Let us see what happen if $baba$ is the input to the NFA $M'$.

- $A \xrightarrow{\lambda} B \xrightarrow{b} \text{crashes} \quad \text{rejected by crashing}$
- $A \xrightarrow{\lambda} D \xrightarrow{b} E \xrightarrow{a} D \xrightarrow{b} E \xrightarrow{a} D \quad \text{accepted}$

Let us also see what happens if $aba$ is the input

- $A \xrightarrow{\lambda} B \xrightarrow{a} C \xrightarrow{b} B \xrightarrow{a} C \quad \text{properly rejected}$
- $A \xrightarrow{\lambda} D \xrightarrow{a} \text{crashes} \quad \text{rejected by crashing}$

Finally let us look at the input $abb$

- $A \xrightarrow{\lambda} B \xrightarrow{a} C \xrightarrow{b} B \xrightarrow{b} \text{crashes, rejected by crashing}$
- $A \xrightarrow{\lambda} D \xrightarrow{a} \text{crashes} \quad \text{rejected by crashing}$

So $M'$ accepts baba and rejects aba & abb.
In other words, a string is accepted if there is at least one way of it being accepted. Observe that the string \( \lambda \) is accepted along two paths & rejected along one path.

\[
\begin{align*}
A & \xrightarrow{\lambda} A \text{ rejected because } A \notin A(M') \\
A & \xrightarrow{\lambda} B \text{ accepted because } B \in A(M') \\
A & \xrightarrow{\lambda} D \text{ accepted because } D \in A(M')
\end{align*}
\]

We will now define precisely what is an NFA and also define what is \( L(M) \) for an NFA \( M \).

**Def.** A Non-deterministic Finite Acceptor (NFA) is a 5-tuple \( M = \langle Q, \Sigma, \Delta, q_0, F \rangle \) where \( Q, \Sigma, q_0 \), and \( F \) are as in a DFA and \( \Delta \subseteq \{Q \times (\Sigma \cup \{\lambda\})\} \times Q \) is a binary relation from \( Q \times (\Sigma \cup \{\lambda\}) \) to \( Q \).

We interpret the ordered pair \( \langle q_1, c \rangle, q_2 \rangle \) as meaning that \( c \) can lead you from \( q_1 \) to \( q_2 \). We sometimes write the ordered pair \( \langle q_1, c \rangle, q_2 \rangle \) as the expression \( q_1 \xrightarrow{c} q_2 \) and call it a **transition**. If \( c = \lambda \), and \( q_1 \xrightarrow{\lambda} q_2 \), we call \( q_1 \xrightarrow{\lambda} q_2 \) a **\( \lambda \)-transition**.

Note also that if the relation \( \Delta \) turns out to be a total function from \( Q \times (\Sigma \cup \{\lambda\}) \) to \( Q \), then the NFA will actually be a DFA. The extended transition relation \( \Delta^* \) is the set of all ordered pairs \( \langle q_1, q_2 \rangle \) such that \( \varphi \) can lead you from \( q_1 \) to \( q_2 \).
Def: Let $q \in T^*$ and $M$ be an NFA. We say that $q$ is accepted by $M$ if we can find a sequence of transitions $q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} q_2 \ldots \xrightarrow{c_n} q_n$ such that $c_1, c_2, \ldots, c_n = q'$ and $q_n \in A(M)$. Here each $c_i \in T \cup \{\lambda\}$. (In other words, $q$ is accepted by $M$ if $\langle q_0, q', q_f \rangle \in \Delta^*$ for some $q_f \in A(M)$.)

The language recognized by an NFA is defined by $L(M) = \{q \in T^* : q$ is accepted by $M\}$.

Def: A One-accepting-state NFA (OAS-NFA) is an NFA which has exactly one accepting state which is different from the initial state.

Prop: Any NFA $M$ is equivalent to an OAS-NFA $M'$.

Proof: Let $Q(M') = Q(M) \cup \{\bar{Z}\}$, where $\bar{Z}$ is a new state that is not already in $Q(M)$. Then add $\lambda$-transitions from each state in $A(M)$ to $\bar{Z}$. Finally let $A(M') = \{\bar{Z}\}$. Then it can be shown that $L(M') = L(M)$. So $M' \cong M$.

Ex.2: Find an OAS-NFA $M'$ which is equivalent to the NFA $M$ shown below.

![Diagram of NFA M and OAS-NFA M']
Ex2. Find an NFA which recognizes the language
\[ L_2 = a b^* b a^* \]

If you try to find a DFA which recognizes \( L_2 \), then you will begin to appreciate how "user friendly" NFAs are.

Q1. Given NFAs \( M_1 \) & \( M_2 \), how can we find NFAs which can recognize:
   (a) \( L(M_1) \cup L(M_2) \)  
   (b) \( L(M_1) \cdot L(M_2) \)  
   (c) \( L(M_1)^* \)  
   (d) \( L(M_1)^R \)  

\[ NFA \text{ for } L(M_1) \cup L(M_2) \]

\[ NFA \text{ for } L(M_1) \cdot L(M_2) \]

\[ NFA \text{ for } L(M_1)^* \]
(d) First convert \( M \) into an OAS-NFA by adding a new accepting state \( \overline{z} \).

\[
\begin{array}{c}
\begin{array}{c}
\text{OAS-NFA} \\
\end{array}
\end{array}
\]

Then make \( \overline{z} \) into the initial state, \( q_0^{(\overline{z})} \) into the only accepting state, and reverse the direction of each transition.

\[
\begin{array}{c}
\begin{array}{c}
\text{NFA for } L(M)^C.
\end{array}
\end{array}
\]

Now if we try to answer Qu.1 with DFAs replacing NFAs, then it will not at all be easy. That is because DFAs are not very easy to combine. But there is one thing that we can do with DFAs that we cannot easily do with NFAs.

**Prop 2:** Let \( M = (Q, \Sigma, \delta, q_0, F) \). Put \( M_c = (Q, \Sigma, \delta, q_0, Q - \delta(M)) \). Then \( L(M_c) = L(M)^c \).

**Proof:** We have \( L(M_c) \iff \delta^*(q_0, w) \notin \delta(M) \)

\[
\iff\delta^*(q_0, w) \notin \delta(M) \iff w \notin L(M).
\]

So \( L(M_c) = L(M)^c \), and we are done.
Ex. 3. Find a DFA $M_c$ such that $L(M_c) = L(M)^c$ where $M$ is the DFA shown below.

\[ M: \]

\[ A \xrightarrow{a} B \xrightarrow{b} C \xrightarrow{b} D \xrightarrow{a} A \]

\[ \]

Ans: $M_c$:

\[ A \xrightarrow{a} B \xrightarrow{b} C \xrightarrow{b} D \xrightarrow{a} A \]

We cannot do this with NFAs because Puppe does not hold for NFAs. Fortunately, we can show in the next section that every NFA is equivalent to a DFA and this will help us.

§ 4. Equivalence of NFAs & DFAs.

Def. Let $M$ be an NFA and $y \in T^*$. The reaching set $R(y)$ of $y$ in $M$ is defined as follows.

$q_n \in R(y)$ if and only if there is a sequence of transitions $q_0 \xrightarrow{c_1} q_1 \xrightarrow{c_2} q_2 \ldots q_{n-1} \xrightarrow{c_n} q_n$ with $c_1 c_2 \ldots c_n = y$ and each $c_i \in \Sigma^*$.

Ex. 4. Let $M_y = \begin{array}{c}
(0) \\
(0,1) \\
(0,1)
\end{array}$.

Then $R(01) = \{ B, C \}$ in $M_y$ because

$A \xrightarrow{0} A \xrightarrow{0} A \xrightarrow{0} B$ & $A \xrightarrow{0} B \xrightarrow{0} C$

Also $R(00) = \{ A, B, C \}$ in $M_y$ because

$A \xrightarrow{0} A \xrightarrow{0} A \xrightarrow{0} A \xrightarrow{0} B, A \xrightarrow{0} B \xrightarrow{0} C$. 
Ex. 5. Let \( M_5 = \) 

\[
\begin{array}{c}
\text{A} \\
\lambda \\
\text{B} \\
\text{C} \\
\text{0} \\
\text{0} \\
1
\end{array}
\]

Then

(a) \( R(\lambda) = \{ A, B \} \) because \( \rightarrow A \frac{1}{3} A \), \( \rightarrow A \frac{7}{8} B \)

(b) \( R(0) = \{ B, C \} \) because \( \rightarrow A \frac{2}{3} B \frac{1}{2} B \), \( \rightarrow A \frac{1}{3} C \)

(c) \( R(00) = \{ B \} \) because \( \rightarrow A \frac{2}{3} B \frac{1}{2} B \)

The \textit{NFA to DFA Algorithm}

\begin{itemize}
  \item \textbf{Input}: An NFA, \( M = (Q, \Sigma, \delta, q_0, F) \)
  \item \textbf{Output}: An equivalent DFA, \( M_D = (Q_D, \Sigma, \delta_D, q_D, F_D) \)
\end{itemize}

1. First find all possible reaching sets by starting with \( R(\lambda) \) and branching out with each character in \( \Sigma \). When a previous reaching set is encountered, that branch is discontinued. This process will give us the reaching set tree.

2. Let \( Q_D = \{ R(\rho) : \rho \in \Sigma^* \} \) be the set of all reaching sets. Initial state of \( M_D = R(\lambda) \) and \( A_D = \{ R(\rho) : R(\rho) \cap F_M \neq \emptyset \} \) be the set of all reaching sets with at least one accepting state of \( M \).

3. The \textit{transition function} \( S : Q_D \times \Sigma \rightarrow Q_D \) is specified by \( S(R(\rho), c) = R(\rho c) \) for each \( c \in \Sigma \), and can be read off from the reaching set tree.

Ex. 6. Let \( M \) be the NFA

\[
\begin{array}{c}
\text{A} \\
\text{0} \\
\text{B} \\
\text{0} \\
\text{1} \\
\text{C} \\
\text{0}
\end{array}
\]

Find the equivalent DFA, \( M_D \).
Ex. 6

\[ R(\lambda) = \{A\} \]

Ex. 7

Let \( M \) be the NFA on the right. Find the equivalent DFA, \( M_D \).

\[ R(\lambda) = \{B, C\} \]

Qu. 2

Given an NFA, \( M \), how can we find an NFA which recognizes the language \( L(M)^c \)?

Hns: First convert \( M \) into an equivalent DFA \( M_D \). Then switch the accepting & non-accepting states of \( M_D \) to get a DFA \((M_D)_c\). This will be your answer because \((M_D)_c\) is also an NFA.
Ex 8 Let M be the NFA on the right. Find an NFA which recognizes the language \( L(M)^c \).

\[ R(\lambda) = \{A\} \]

\[ \emptyset \]

Final answer = \( (M_D)^c \)

\[ M_D = \]

**Theorem 3**: Let \( M = (Q, \Sigma, \delta, q_0, A) \) be any NFA and \( M_D = (Q_D, \Sigma, \delta, R(\lambda), A_D) \) be the DFA obtained from the NFA to DFA algorithm. Then \( L(M_D) = L(M) \).

**Proof**:

\[ \iff w \in L(M_D) \iff w \text{ can lead you from } R(\lambda) \text{ in } M_D \text{ to an accepting state } R_A \text{ in } A_D \]

\[ \iff w \text{ can lead you from } q_0 \text{ in } M \text{ to an accepting state } q_A \text{ in } A \text{ (because } R(\lambda) = \text{ all states you can reach from } q_0 \text{ using } \lambda), \text{ and } R_A \text{ is an accepting state of } M_D \text{ if and only if } R_A \text{ contains at least one accepting state } q_A \text{ of } M. \]

\[ \iff w \in L(M). \]

So \( L(M_D) = L(M) \).

End of Ch. 3.