1 Penta NFA

A penta NFA is an NFA that accepts a word \( w \) if there exist computation paths for \( w \) such that one-fifth or more of the ending states are accepting. Show that penta NFAs and DFAs are equivalent.

Solution:

1. Consider any input to a DFA. Since the machine is deterministic, there is only one path that the machine takes on any input and hence the machine ends in only one state. If this state is an accepting state, then it trivially means that more than one-fifth of the ending states are accepting and if this state is not an accepting state, then it trivially means that less than one-fifth of the ending states are accepting. Thus, a DFA is automatically a penta NFA.

2. Intuition:

Given a penta NFA, we want to show that we can construct a DFA. Let’s use the NFA→DFA conversion procedure to change the NFA to a deterministic finite automaton. Therefore, the states of the new DFA is the power set of all the states of the NFA. In that transformation, a state in the DFA (which is a subset of the states of the original NFA) is an accepting state only if one of the states in the subset was an accepting state in the original NFA.

However, to satisfy the penta-NFA constraint, we mark a state in the DFA as accepting only when at least one-fifth of the states in the subset are accepting states in the original penta NFA. We keep the rest of the construction (i.e the alphabet, starting state and transition function) the same.
Construction
Given penta NFA $M = (Q, \Sigma, \delta, q_0, F)$, we can construct a DFA $M' = (Q', \Sigma, \delta', q'_0, F')$.
(This construction was covered in class) Let $P(Q)$ denote the power set of the set $Q$.

$Q' = P(Q) \quad q'_0 = q_0$

$\forall S \in P(Q), \delta'(S, a) = \bigcup_{q \in \epsilon \text{-closure}(S)} \epsilon \text{-closure}(\delta(q, a))$

$F' = \{ S \in P(Q) \mid \frac{|S \cap F|}{|S|} \geq \frac{1}{5} \}$

Let’s examine why this construction is correct. Consider any input that is accepted by the DFA. Suppose it ends up in an accepting state $S$. When we look at the subset denoted by $S$, since $S$ is an accepting state, at least one-fifth of the elements in this subset must be accepting states in the original penta NFA $M$. Therefore, this string would also be accepted by the penta NFA. Similarly we can see that any string not accepted by the DFA would also not be accepted by the penta NFA.
Let $L$ be any language, and let $L_{alt}$ be the set of string in $L$ with every other character removed, i.e.

$$L_{alt} = \{ x \mid \exists y \in L \text{ such that } x_1x_2x_3\ldots = y_1y_3y_5\ldots \}$$

Show that if $L$ is a regular language, then $L_{alt}$ is regular.

**Solution:**

**Intuition**
Consider a regular language $L$ and let $M$ be any DFA which accepts $L$. For the new language $L_{alt}$, we would like to use a NFA to guess for us every other symbol when trying to accept a string $x$. We will need to do this in a manner such that $x_1x_2x_3\ldots = y_1y_3y_5\ldots$ and $y \in L$. To achieve this let’s use two copies of $M$ to construct $M'$. Let’s call these copies as $M_1$ and $M_2$. The set of states of the new machine will be the union of the states of both the copies. The idea is that whenever we see an input symbol, we will start in a state of machine $M_1$, process that symbol and move to a state in $M_2$. In order to achieve this, suppose the machine on seeing an input symbol moved to a state $q$ in $M_1$. We will delete that transition from our new machine and add a transition to the equivalent state $q_1'$ in the other copy $M_2$. Therefore, on seeing that input symbol, the machine does almost the same thing as the original machine, but just moves to the equivalent state in the second copy. Then, from $M_2$, we will make an epsilon transition to a state in $M_1$ in order to mimic the missing symbol. That is, from the current state in $M_2$, suppose we would transition to a state $q_2'$ on seeing some symbol $c$. Now, the new machine will delete this transition and even without seeing symbol $c$, will make an epsilon transition to the equivalent state of $q_1'$ in the first copy $M_1$.

In this manner, we keep moving alternately between the two machines - using the first machine to process the actual input and the second machine to process the alternating missing input. We will start at the initial state of $M_1$. The accepting states of the new machine will be the union of accepting states of the two copies.

**Construction**
Let $M$ be a DFA which accepts $L$. Let’s create a new machine $M'$ that has two copies of $M$ : $M_1$ and $M_2$ and let the states of $M'$ be $Q' = Q_1 \cup Q_2$.

The transition function is defined as follows. Suppose we are in state $q^1_k \in M_1$ and $\delta(q^1_k, a) = q^1_j$. We take this transition out and add the following new transition $\delta(q^1_k, a) = q^2_j$. We move to the state $q^2_j \in M_2$. Furthermore, consider any transition $\delta(q^2_j, a) = q^2_k$ in $M_2$. We modify this transition as follows. We replace $a$ by $\epsilon$ (to represent a skipped character) and we replace $q^2_k$ by $q^1_k$ so that we come back to the machine $M_1$. Thus, we remove this transition and add the transition $\delta(q^2_j, \epsilon) = q^1_k$. We make this transformation on all the transitions of the machines $M_1$ and $M_2$ to get $M'$. The start state of the new machine is the start state.
of $M_1$. The final states will be the original final states of $M_1$ and the original final states of $M_2$ (to capture the cases when $y$ has odd length.)

To see that this new NFA accepts the language $L_{alt}$, observe that for every string $y_1y_2y_3\ldots \in L$ there is an accepting path for the string $y_1\epsilon y_3\epsilon y_5\ldots \in L_{alt}$ that moves from machine $M_1$ to $M_2$ on every $y_i$, where $i$ is odd, and then transitions back to machine $M_1$ using $\epsilon$ transitions. By a similar argument, we can see why any accepting string in $L_{alt}$ should have a corresponding string (when we fill in the skipped characters) in $L$. Hence, $L_{alt}$ is regular.
3 \( L_n \)

(Problem 1.41 in textbook)
Show that if \( L \) is a regular language, then
\[
L_n = \{0^k \mid k \text{ is a multiple of } n\}
\]
is regular.

Solution:

Let \( n \) be any integer greater than or equal to 0.
Let \( M = (Q, \Sigma, \delta, q_0, F) \) be a DFA such that
\[
Q = \{q_0, q_1, \ldots, q_{n-1}\}
\]
\[
F = \{q_0\}
\]
\[
\delta(q, a) = \begin{cases} 
\{q_{i+1}\} & \text{if } q = q_i \text{ for some } 0 \leq i < n - 1 \\
\{q_0\} & \text{if } q = q_{n-1}
\end{cases}
\]

Given our construction, we can see that \( M \) consists of a set of \( n \) states arranged in a loop. Now, since \( q_0 \) is the only accepting state, \( M \) accepts \( x \) if and only if \( x \) is a string that causes \( M \) to make a complete loop zero or more times. (i.e. \( M \) goes from \( q_0 \) to \( q_1 \) to \( q_2 \) to \ldots to \( q_{n-1} \) to \( q_0 \) one or more times). If \( M \) makes only a partial loop, then \( M \) will not end in an accepting state. Now, \( M \) goes around the loop one time for each string \( 0^n \). If it sees any characters other than 0, \( M \) will halt that branch of computation. Thus, \( M \) accepts exactly those strings which equal \( (0^n)^a \) for some \( a \in \mathbb{N} \). But this means that \( M \) accepts exactly those strings of the form \( 0^k \) for some \( k \) which is a multiple of \( n \). Thus, \( L(M) = L_n \).
4 \textit{NOPREFIX} \noprefl (Problem 1.45a in textbook)

Show that if \( L \) is a regular language, then

\[
\text{NOPREFIX}(L) = \{ w \in L \mid \text{no proper prefix of } w \text{ is a member of } L \}
\]

is regular.

\textbf{Solution:}

\textbf{Intuition:}
If \( x \) is a proper prefix of \( w \), then \( w = xy \) for some \( y, |y| > 0 \).

Now, we want to have the same language as before, except that we eliminate all \( w \) in \( L \) which have a proper prefix \( x \) that is also a member of \( L \).

Suppose that \( M \) is a DFA for \( L \). Then, if \( w \) and \( x \) are both members of \( L \), we have the following case:

\begin{center}
\begin{tikzpicture}[node distance=2cm,auto,>=latex]
  \node[state] (q0) {};
  \node[state] (q1) [right of=q0] {};
  \node[state] (q2) [right of=q1] {};
  \path[->] (q0) edge node {$x$} (q1);
  \path[->] (q1) edge node {$y$} (q2);
\end{tikzpicture}
\end{center}

To eliminate these \( w \), we want to get rid of the case where we can reach an accepting state after another accepting state:

\begin{center}
\begin{tikzpicture}[node distance=2cm,auto,>=latex]
  \node[state] (q0) {};
  \node[state] (q1) [right of=q0] {};
  \node[state] (q2) [right of=q1] {};
  \path[->] (q0) edge node {$x$} (q1);
  \path[->] (q1) edge node {$y$} (q2);
  \path[->] (q1) edge[red,loop] (q1);
\end{tikzpicture}
\end{center}

We can do this by getting rid of all transitions out of accepting states. To see why this works, suppose for a string \( w \in L \), \( M \) goes past an accepting state. If on input \( w \), \( M \) does not end up in an accepting state, then \( M \) rejects \( w \). However, if on input \( w \), \( M \) ends up in an accepting state, then \( M \) should also reject because this means that \( w \) has a proper prefix which is a member of \( L \) (corresponding to the path from the start state to the first accepting state). So, we see that once \( M \) goes past an accepting state, \( M \) will always reject. Thus, we can get rid of transitions out of accepting states, causing \( M \) to reject on default if there is additional input.

\textbf{Construction:}
Let \( L \) be a regular language. Then, there exists a DFA \( M = (Q, \Sigma, \delta, q_0, F) \) such that \( L(M) = L \).

Let \( M' = (Q', \Sigma, \delta', q'_0, F') \) be a DFA such that \( Q' = Q \).
\[
q'_0 = q_0 \\
F' = F \\
\delta'(q, a) = \begin{cases} 
\delta(q, a) & \text{if } q \notin F \\
\emptyset & \text{if } q \in F
\end{cases}
\]

Now, if \( w \in NOPREFIX(L) \), then \( w \) is in \( L \), so \( M(x) \) accepts. Thus, on input \( w \), \( M \) goes from \( q_0 \) to some state \( q'' \in F \). If this path goes through any accepting states along the way, then \( w \notin NOPREFIX(L) \) since there is a proper prefix of \( w \) which is in \( L \) (corresponding to the path from the start state to this accepting state). Thus, \( M \) does not pass through any accepting states between \( q_0 \) and \( q'' \). Since \( M' \) only differs from \( M \) when it is transitioning out of accepting states and since \( M \) does not transition out of any accepting states along this path, then \( M' \) will act the same as \( M \) on input \( w \). Thus, \( M'(w) \) accepts.

If \( w \notin NOPREFIX(L) \), then either \( w \) is not in \( L \), or \( w \) has a proper prefix which is also in \( L \). Suppose that \( w \) is not in \( L \). Now, \( M' \) is the same as \( M \) except that some transitions have been deleted. Thus, \( M' \) acts the same as \( M \) except that it cuts off some of the branches of computation. Therefore, \( M' \) can never accept more strings than \( M \). Therefore, since \( M \) does not accept \( w \), then \( M' \) will not accept \( w \). Now, if \( w \) has a proper prefix \( x \) which is also in \( L \), then \( w = xy \) for some \( y, |y| > 0 \). So, \( M \) will go from \( q_0 \) to \( q_a \in F \) on input \( x \), and from \( q_a \) to \( q_b \in F \) on input \( y \). But, \( M' \) acts the same as \( M \) except that it never transitions out of an accepting state. Thus, \( M' \) will not transition out of \( q_a \) and will halt that branch of computation. Therefore, \( M' \) will reject \( x \).
5 \textit{L}_1 \textit{ avoids L}_2

(Problem 1.67 in textbook)
Show that if \textit{L}_1, \textit{L}_2 are regular then
\[\textit{L}_1 \textit{ avoids L}_2 = \{w \mid w \in \textit{L}_1 \text{ and } w \text{ doesn't contain any string in } \textit{L}_2 \text{ as a substring}\}\]
is regular.

\textbf{Solution:}

\textbf{Intuition}
Let \textit{M}_1 and \textit{M}_2 accept \textit{L}_1 and \textit{L}_2 respectively. We will construct a new NFA \textit{N}.
To track whether \textit{w} is in \textit{L}_1, we will run \textit{w} on \textit{M}_1. Concurrently, we will also keep track of all possible substrings of \textit{w} in \textit{L}_2 by running all possible substrings of \textit{w} on \textit{M}_2. If one of the substrings of \textit{w} is accepted in \textit{M}_2, then we will stop that branch of computation and reject. To do so, we will take the set of states in \textit{N} to be the Cartesian product of \textit{Q}_1, representing the state that \textit{w} is in in \textit{M}_1, and \textit{P}(\textit{Q}_2), representing all the states that substrings of \textit{w} (that start at any point processed so far) could be in in \textit{M}_2. At each step, we will transition in \textit{M}_1 as normal by using \textit{δ}_1. We will then transition each of the substrings in \textit{M}_2 by 1 step by using \textit{δ}_2. We will also start a new substring beginning at the current point in \textit{w} by adding the start state of \textit{M}_2 to the set of current substring states.

\textbf{Construction}
Let \textit{L}_1 and \textit{L}_2 be regular languages.
Then, there exists DFAs \textit{M}_1 = (\textit{Q}_1, \Sigma, \textit{δ}_1, q_1, F_1) and \textit{M}_2 = (\textit{Q}_2, \Sigma, \textit{δ}_2, q_2, F_2) such that \textit{L}(\textit{M}_1) = \textit{L}_1 and \textit{L}(\textit{M}_2) = \textit{L}_2.
Let \textit{N} = (\textit{Q}', \Sigma, \textit{δ}', q'_0, F') be an NFA such that
\[\textit{Q}' = \textit{Q}_1 \times \textit{P}(\textit{Q}_2)\]
\[q'_0 = (q_1, \{q_2\})\]
\[F' = \{(q, S) \mid q \in \textit{F}_1 \text{ and } \forall s \in S, s \notin \textit{F}_2\}\]
For \(q \in \textit{Q}_1, S \in \textit{P}(\textit{Q}_2), a \in \Sigma,\)
\[\textit{δ}'((q, S), a) = \begin{cases} (\textit{δ}_1(q, a), \{\textit{δ}_2(s, a) \mid s \in S\} \cup \{q_2\}) & \text{if } \forall s \in S, s \notin \textit{F}_2 \\ \emptyset & \text{else (i.e. } \exists s \in S \text{ s.t. } s \in \textit{F}_2\)\]
If $w \notin L_1$ avoids $L_2$, then either $M_1(w)$ rejects, or $M_2(x)$ accepts for some $x$ that is a substring of $w$. Suppose that $M_1(w)$ rejects. Now, consider the states $(q'_0, S_0), (q'_1, S_1), (q'_2, S_2), \ldots, (q'_n, S_n)$ that $N$ goes through on input $w$. Now, $q'_i$ either transitions according to the transition function of $M_1$ or that branch of computation ends. Thus, $q_n$ will either be the state that $w$ would have ended up in $M_1$, or will be the last state before that branch of computation dies. Since $M_1(w)$ rejects, then $q'_n \notin F_1$, so $(q'_n, S_n) \notin F'$. So, either way, $N$ will reject $w$. Suppose otherwise, then, that $M_1(w)$ accepts, but that there is some substring $x$ of $w$ such that $x \in L_2$. Suppose this substring ended at position $j$ in $w$. Then, $S'_j$ will contain the state that $M_2$ would have ended up in on input $x$. Since $M_2(x)$ accepts, then $S'_j$ will contain an accepting state of $M_2$. But this means that either the branch of computation dies after the next transition, or that the current state of $N$ is not an accepting state. Thus, $N$ will not accept $w$. 