Finite Automata and Regular Expressions - I

A finite automaton has a finite set $Q$ of states. Its input is a string of letters from a finite alphabet $\Sigma$. It starts at time 0 in a designated start state $q_0 \in Q$ at the left end of the input string. At time $i$ it reads the $i$ th letter of the input string and changes state according to a “transition function” $\delta : Q \times \Sigma \rightarrow Q$. The set of states $Q$ is partitioned into two parts - $F, Q \setminus F$, where $F$ are the “final” or accepting states. If at the end, it is in one of the states in $F$, it “accepts” the string, otherwise it “rejects” the string. The set of all strings (over $\Sigma$) accepted by the automaton is called the “language” accepted by it.

$\Sigma^*$ denotes the set of all finite length strings over $\Sigma$ and a subset of $\Sigma^*$ is said to be “regular” if and only if it is accepted by some finite automaton.

The above is called a deterministic finite automaton - DFA. A non-deterministic finite automaton - NFA - has the property that it can non-deterministically choose one of several possible next states - i.e., $\delta$ now is a function from $Q \times \Sigma$ to $2^Q$. We say that an NFA accepts a string iff there is some valid computation of it on the string which leads to an accepting state.

**Theorem** (Equivalence of NFA and DFA) : For any language $L$ accepted by a NFA, there is a DFA which accepts $L$.

The proof gives a DFA which “simulates” the NFA accepting $L$. The DFA just “remembers” all possible states the NFA could have been in at each time; for this the set of states of the DFA just needs to be $2^Q$, still finite.

DFA as well as NFA can be represented by graphs.

$\epsilon$ - moves : NFA with $\epsilon$ moves can make transitions labelled $\epsilon$ without using up any letter of the input.

**Theorem** A language accepted by an NFA with $\epsilon$-moves can be accepted by an NFA without $\epsilon$ moves.

**Regular Expressions** represent languages. The regular expressions over $\Sigma$ and the sets they represent are defined recursively as follows:

1. $\phi$ is a regular expression and denotes the empty set.
2. $\epsilon$ is a regular expression and denotes the set $\{\epsilon\}$.
3. For each $a \in \Sigma$, $a$ is a regular expression denoting the set $\{a\}$. 


4. If \( r, s \) are regular expressions, denoting sets \( R, S \) respectively, then 
\((r + s), (rs)\) and \( (r^*) \) are regular expressions denoting sets \( R \cup S \), 
\( RS \) (concatenation) and \( R^* \) respectively.

For a regular expression \( r \), the set represented by it is denoted \( L(r) \).

**Theorem** Let \( r \) be a regular expression. Then there exists an NFA with 
\( \epsilon \)-transitions accepting \( L(r) \).

**Theorem** If \( L \) is accepted by a DFA, then \( L \) can be represented by a 
regular expression.