Closure Properties of Regular Languages

- **Definition:** A set $S$ is said to be closed under some operation $O_p$, if, whenever $O_p$ is applied to the members of $S$, the result is also a member of $S$.

- **Theorem:** The class of regular languages is closed under complement. That is, if $L$ is a regular language, the so is $\overline{L}$.

  **Proof** Let $L$ be a regular language over some alphabet $\Sigma$. Since $L$ is regular, there exists a DFA $M = (Q, \Sigma, \delta, q_0, F)$ which recognizes $L$. To prove that $\overline{L}$ is regular, we will construct a DFA $M'$ that recognizes $\overline{L}$.

  The idea: Take $M'$ to be exactly that same as $M$, except the set of accepting states for $M'$ is $Q - F$.

  To prove correctness show that $w \in L$ if and only if $M'$ accepts $w$. For this we observe that since the set of states, the initial state and the transition function of $M$ and $M'$ are exactly the same, $(r_0, r_1, \ldots, r_n)$ is a computation of $M$ on $w$ if and only if $(r_0, r_1, \ldots, r_n)$ is a computation of $M'$ on $w$.

  Now, take $w \in L$ . . . 

  **Exercise:** Complete the details of the above argument.

- **Theorem:** The class of regular languages is closed under union and intersection. That is, if $L_1$ and $L_2$ are regular languages, the so are $L_1 \cup L_2$ and $L_1 \cap L_2$.

  **Proof** Let $L_1$ and $L_2$ are languages over the same alphabet $\Sigma$. Since $L_1$ is regular, there exists a DFA $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ which recognizes $L_1$. Similarly, there exists a DFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ which recognizes $L_2$.

  To prove that $L_1 \cup L_2$ is regular, we will construct a DFA $M_{\cup}$ which recognizes $L_1 \cup L_2 = \{w | w \in L_1 \text{ or } w \in L_2\}$.

  The idea: $M_{\cup} = (Q, \Sigma, \delta, q_0, F)$ simulates a parallel execution of $M_1$ and $M_2$. $M_{\cup}$ is defined as follows:

  - $Q = Q_1 \times Q_2$;
  - $\Sigma$ is the same;
  - $\delta((q_1, q_2), a) = (\delta(q_1, a), \delta(q_2, a))$;
  - $q_0 = (q_1, q_2)$;
  - $F = \{(r_1, r_2) | r_1 \in F_1 \text{ or } r_2 \in F_2\}$
To prove correctness we need to show that $w \in L_1 \cup L_2$ if and only if $M_\cup$ accepts $w$. This, in turn, follows from the fact that $(r_0, r_1, \ldots, r_n)$ is a computation of $M_1$ on $w$, and $(t_0, t_1, \ldots, t_n)$ is a computation of $M_2$ on $w$, if and only if $((r_0, t_0), (r_1, t_1), \ldots, (r_n, t_n))$ is a computation of $M_\cup$ on $w$.

\textbf{Exercise:} Complete the details of the above argument (\textit{Hint:} use induction).

\textbf{Exercise:} What would the DFA that accepts $L_1 \cap L_2$ look like ?

- \textbf{Corollary:} Every finite language is regular.

- \textbf{Theorem:} The class of regular languages is closed under concatenation and Kleene’s star operation. That is, if $L_1$ and $L_2$ are regular languages, the so are $L_1 L_2$ and $L_1^*$. The proof based on the idea of “concatenation” of two DFAs will not fly (Why ?). The proof of this fact will require some new concepts.
Non-deterministic Finite Automata

• A non-deterministic finite automaton (NFA) is a generalization of DFA with the following two changes:

  – In DFA for each pair \((q, a)\) with \(q \in Q\), \(q \in \Sigma\), the transition function \(\delta\) gave exactly one next state. In NFA, for each pair \((q, a)\) the transition function may give 0, 1, or more than one next states.

    * 0 next states – \(\delta(q, a)\) is not defined.
      In this case, if the NFA is in state \(q\) and receives input symbol \(a\), it will reject the rest of the input, regardless of what it is (it “dies”). This is similar to going into a trap state.

    * 1 next state – \(\delta(q, a) = r\).
      In this case, the NFA in state \(q\) on input symbol \(a\) will behave just like a DFA – it will transition to state \(r\).

    * Multiple next states – \(\delta(q, a) = \{r_1, r_2, ..., r_n\}\).
      In this case, the NFA in state \(q\) on input symbol \(a\) splits itself into \(n\) copies. Copy \(i\) goes into the state \(q_i\). All the copies are now run “in parallel” to each other.

In this NFA, the transition \(\delta(q_0, 1)\) is not defined. Thus, every string that starts with 1 will “kill” the NFA immediately, and so will be rejected.

In this case, the NFA in state \(q\) on input symbol \(a\) will behave just like a DFA – it will transition to state \(r\).

In this case, the NFA in state \(q\) on input symbol \(a\) splits itself into \(n\) copies. Copy \(i\) goes into the state \(q_i\). All the copies are now run “in parallel” to each other.
– In DFA a transition from state state occurs only when a new input symbol is consumed. In NFA, some transitions (called \( \varepsilon \)-transitions) occur without consuming any input symbols.