Week Four: Regular Languages; Pumping Lemma

1. There are other methods of finding a regular expression equivalent to a finite automaton in addition to the ones discussed in the last lecture. One treats the problem as one of solving a system of equations for the language, where concatenation plays the role of multiplication and union the role of addition.

We are given the DFA $M = (Q, \Sigma, \delta, s, F)$. Let $X_q = \{ x \in \Sigma^* : \delta(q, x) \in F \}$ which is the set of all strings in $\Sigma^*$ that can accepted by the DFA if its start state is $q$ rather than $s$.

Then we find that:

$$X_q = \bigcup_{a \in \Sigma} aX_{\delta(q, a)} : \text{if } q \notin F, \text{while } = \bigcup_{a \in \Sigma} aX_{\delta(q, a)} \cup \epsilon, \text{if } q \in F.$$

This is the linear system for the sets $X_q$'s we mentioned above where addition is replaced by union and multiplication by concatenation.

For regular languages we need what is known as Arden's Lemma:

Arden's Lemma: Let $A, B \subset \Sigma^*$ with $\epsilon \notin A$. Then the equation: $X = A \circ X \cup B$ has the unique solution $X = A^* \circ B$.

Proof: Step 1: If $X$ is a solution then $A^*B \subset X$.

To see this, note that

$$A^*B = (A^+ \cup \epsilon)B = A^+B \cup B = A(A^*B) \cup B.$$

Step 2: $X \subset A^*B$.

By Step 1 $X = A^*B \cup C$, since $A^*B \subset X$ with $C \cap A^*B = \emptyset$. We want to show that $C = \emptyset$.

Now $X = AX \cup B$, so

$$A^*B \cup C = A(A^*B \cup C) \cup B = A^+B \cup AC \cup B$$

$$= A^+B \cup B \cup AC = (A^+ \cup \epsilon)B \cup AC = A^*B \cup AC$$
We conclude: $A^*B$ is the unique solution.

**Note:** If $e \in A$, then the solution $A^*B$ is no longer unique but it is the smallest solution.

**Example:** We shall indicate an outline of the calculation of the regular expression for the DFA given in the last lecture using the equational approach.

We obtain the following system of equations:

\[
X_1 = aX_2 \cup bX_3 \\
X_2 = aX_1 \cup bX_3 \cup \epsilon \\
X_3 = aX_2 \cup bX_3 \cup \epsilon
\]

Note the term $\epsilon$ in the equations for the states which are accepting. The goal is to find $X_1$ since $q_1$ is the initial state.

We start solving this system. Since $X_1 = aX_2 \cup bX_3$ we obtain $X_2 = a(aX_2 \cup bX_3) \cup bX_3 \cup \epsilon$.

By Arden's Lemma we find $X_2 = (aa)^*[(ab \cup b)X_3 \cup \epsilon]$. By substituting this equation into $X_3 = aX_2 \cup bX_3 \cup \epsilon$ we obtain another equation for $X_3$ that can be solved using Arden's Lemma. Because $X_2$ and $X_3$ are now known these identities can be used in $X_1 = aX_2 \cup bX_3$ to determine $X_1$.

2. We now present a useful theoretical result that states that regular languages must obey a certain type of "periodicity" property. It is used to show that certain simple languages cannot be regular.

**Pumping Lemma.** Let $M = (K, \Sigma, \delta, s, F)$ be a DFA with $L = L(M)$. Suppose $m = |K|$. Let $w \in L(M)$ with $|w| \geq m$. Then there are strings $x, y, z$ such that $w = xyz$, $|xy| \leq m$, $y \neq \epsilon$, and $xy^kz \in L$, $\forall k \geq 0$. We call $m$, the **pumping constant**.

**Idea of the Proof.** Any string accepted by $M$ whose length is greater than the number of states of the machine must have a loop in it. It is precisely this loop that can be iterated.
Week Four .......................... Regular Languages; Pumping Lemma

We use the contrapositive form of the pumping lemma to show that a language is NOT regular.

Let \( L \) be a language. Suppose that there exists a string \( w \) with substrings \( x, y, z \) such that \( y \neq \epsilon, w = xyz, \) and \( xy^kz \notin L, \) for some integer \( k \geq 0, \) then \( L \) cannot be a regular language.

So, to show a language is NOT regular, think of playing the following sort of game: find a string \( w \in L \) so that for any non-empty substring \( y \) of \( w, \) there exists some pumped form of \( w : xy^kz \) so that \( xy^kz \notin L. \)

**Example (1)** \( L_1 = \{a^n b^n : n \geq 1\} \) is not regular.

Suppose the language were regular. Choose \( n \) greater than the pumping constant given above. Then \( w = a^n b^n \) can be factored as \( xyz, y \neq \epsilon, \) and \( xy^kz \in L_1, \) for all \( k \geq 0. \) Choose \( k = 0, \) so \( xz \in L, \) but \( xz = a^n b^n \notin L. \) Contradiction. We say that we pumped "down" in this example.

**Example (2)** \( L_2 = \{a^{n^2} : n \geq 1\} \) is not regular.

Suppose \( L_2 \) were regular. Choose \( n \) greater than the pumping constant \( m. \) Then \( a^{n^2} = xyz, \) where \( y \neq \epsilon \) and \( |y| \leq m \leq n. \) So \( xy^kz \in L_2, \) for all \( k \geq 0. \) Choose \( k = 2. \) Then \( xy^2z \in L_2 \) implies \( |xy^2z| \) is a perfect square. But \( n^2 < |xy^2z| < n^2 + n < (n+1)^2, \) contradiction. In this example we say that we pumped "up."

**Example (3)** \( L_3 = \{w \circ w^R : w \in \Sigma^*, \Sigma = \{a, b\}\} \) is not regular if \( \Sigma = \{a, b\}. \)

Suppose \( L_3 \) were regular. Choose the string \( w \) so \( |w| \geq m + 1, \) where \( m \) is the pumping constant. Further we may choose \( w \) to have the special form: \( w = a^m b, \) so \( ww^R = a^m bba^m. \) By the Pumping Lemma \( w w^R = xyz, \) with \( |xy| \leq m \) and \( xy^kz \in L_3, \) for \( k \geq 0. \) Take \( k = 0. \) Then \( xz = a^m bba^m \in L_3. \) Contradiction.

3. **Problem:** Given a DFA \( M, \) find an equivalent DFA with a minimum number of states.
We present two solutions to this problem. The first one is algorithmic. The second one is more conceptual and proves that the equivalent minimum state DFA is unique up to the labeling of its states.

4. First Method: Merging of Equivalent States

Let $M = (K, \Sigma, \delta, q_0, F)$ be a DFA. Given two states $q$ and $q'$ from $K$, we define an equivalence relation $\sim$ on the states of $M$ by:

$q \sim q'$ means $\delta(q, w) \in F \iff \delta(q', w) \in F, \ \forall w \in \Sigma^*.$

The $\sim$-equivalence classes are computed by a sequence of other equivalence relations $\equiv_n$ by successive refinements.

Let $q$ and $q'$ be two states of $M$. Then: $q \equiv_0 q'$ means $q \in F \iff q' \in F$. That is $\Gamma \equiv_0$ has two equivalence classes: the set of accepting states $F$ and the set of rejecting states $Q \setminus F$.

For $n > 0$, define $\equiv_{n+1}$ to mean: $q \equiv_{n+1} q'$ as $q \equiv_n q'$ and $\delta(q, a) \equiv_n \delta(q', a), \ \forall a \in \Sigma$. That is $\Gamma q \equiv_n q'$ means $\delta(q, w) \in F \iff \delta(q', w) \in F$, for all strings $w$ whose length is less than or equal $n$.

The equivalence classes of $\equiv_n$ stabilize for $n$ less than or equal to the number of states of the automaton $M$. Further $\Gamma q \sim q'$ if and only if $q \equiv_n q'$ for all $n$. Now if all the states of $M$ are reachable from the start state $q_0$ and if equivalent states are merged then the resulting automaton has a minimum number of states.

These observations give rise to an effective algorithm to find the minimum state automaton by successively computing the $\equiv_n$-equivalence classes $\Gamma$ for $n = 0, 1, 2, \ldots$. The process terminates when the equivalence classes for two successive values of $n$ agree.

Algorithm for Merging Equivalent States:

We first make a table of unordered pairs of distinct states. No pair is marked.

(1) First mark all pairs of inequivalent states relative to strings of length 0; so mark the pair $\{p, q\}$ if $p \in F, q \in Q \setminus F$ or $p \in Q \setminus F, q \in F$. 

4
(2) Next we mark all pairs of inequivalent states relative to strings of length $k = 1, 2, \ldots, n$, where $n$ is the total number of states of the original DFA.

for $k = 1 \ldots n$ do

if there is an unmarked pair $\{p, q\}$ so that $\{\delta(p, \sigma), \delta(q, \sigma)\}$ is marked then mark the pair $\{p, q\}$. od;

(3) When the loop terminates all inequivalent pairs are marked; so the unmarked pairs are equivalent states. Merge these pairs together.

Example.

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

The accepting states are $\{1, 2, 5\}$.

The result of the algorithm is seen to be that states 1 and 2 should be merged and states 3 and 4 should be merged as well.

\[
\{0,3\} \xrightarrow{a} \{1,5\}, \quad \{0,3\} \xrightarrow{b} \{2,5\}.
\]
\[
\{0,4\} \xrightarrow{a} \{1,5\}, \quad \{0,4\} \xrightarrow{b} \{2,5\}
\]
\[
\{1,2\} \xrightarrow{a} \{3,4\}, \quad \{1,2\} \xrightarrow{b} \{3,4\}.
\]
\[
\{1,5\} \xrightarrow{a} \{3,5\}, \quad \{1,5\} \xrightarrow{b} \{4,5\}.
\]
\[
\{3,4\} \xrightarrow{a} \{5,5\}, \quad \{3,4\} \xrightarrow{b} \{5,5\}.
\]
\[
\{2,5\} \xrightarrow{a} \{4,5\}, \quad \{2,5\} \xrightarrow{b} \{3,5\}.
\]
5. Second Method: Construction of the Minimum State DFA directly from the Language $L$

Let $M = (K, \Sigma, \delta, q_0, F)$ be a finite deterministic automaton such that all its states are reachable from its start state. Let $L = L(M)$ be the language it accepts.

We associate with $M$ a special equivalence relation $R_M$ on $\Sigma^*$, where

$$xR_My \iff \delta(q_0, x) = \delta(q_0, y),$$

where $x, y \in \Sigma^*$ that is, two strings $x$ and $y$ are equivalent if they terminate at the same state. Hence we can identify the $R_M$ equivalence classes $[x]_M$ with the states of $M$.

The language $L(M)$ is the union of the $R_M$-equivalence classes which include an element $x$, so $\delta(q_0, x) \in F$. We may call $R_M$, machine equivalence.

We call an equivalence relation $R$ on $\Sigma^*$ right-invariant if $xRy \Rightarrow xzRyz$, for all strings $z \in \Sigma^*$.

Note: $R_M$ is right invariant.

Let $L$ be any language over the alphabet $\Sigma$, that is $L \subseteq \Sigma^*$. We can associate an equivalence relation $R_L$ on $\Sigma^*$ directly from $L$, without using a finite automaton.

Given any two strings $x, y \in \Sigma^*$, we say $xR_Ly \iff xz \in L$ exactly when $yz \in L$, for all $z \in \Sigma^*$.

Note: the equivalence relation $R_L$ is right invariant and $R_L$ is a refinement of $R_M$, if $L = L(M)$, for a DFA $M$.

6. We can construct a deterministic finite automaton $M_L$ directly from the equivalence relation $R_L$, if $R_L$ has FINITE index; that is if the number of $R_L$ equivalence classes is finite.

We set $M_L = (K_L, \Sigma, \delta_L, s_L, F_L)$.

Let $K_L$, the states of the machine $M_L$, be the collection of all $R_L$-equivalence classes; write them as $[x]_L$, for a string $x$. 

6
The transition function $\delta_L : K \times \Sigma \to K$ is given as: $\delta_L([x]_L, a) = [xa]_L$. Note: $\delta_L$ is well defined.

Set $s_L = [\epsilon]_L$ and $F_L = \{[x]_L : x \in L\}$.

The minimum state automaton accepting $L$ is given by $M_L$, further any other minimum state automaton that accepts $L$ can be identified with $M_L$, by a re-labelling of its states.

The regular languages are closed under homomorphisms.

A homomorphism is a map $h : \Sigma^* \to \Gamma^*$ such that for all $x, y \in \Sigma^*$ we have that

$$h(xy) = h(x)h(y) \text{ and } h(\epsilon) = \epsilon.$$  

It follows at once that the values of $h$ are determined on any string once they are known for the letters of $\Sigma^*$.

**Proposition.** Let $h : \Sigma^* \to \Gamma$ be a homomorphism. Let $L \subseteq \Sigma^*$ be a regular language. Then $h(L)$ is also regular.

For the proof we use the fact that $L$ is denoted by some regular expression. Then the argument reduces to the following formula:

$$L(h(\beta)) = h(L(\beta)),$$

for any regular expression $\beta$. This formula can be established by induction on the number of operators in the regular expression $\beta$.

The base case of zero operators corresponds to $\beta = \sigma \in \Sigma$, $\epsilon$ or $\emptyset$. In all three cases the desired formula holds.

We next need to make the observations that

1. $h(L_1 \circ L_2) = h(L_1) \circ h(L_2)$
2. $h(\bigcup L_k) = \bigcup h(L_k)$

The result now follows in a routine fashion.
Proposition. Let \( h : \Sigma^* \to \Gamma^* \) be a homomorphism. Let \( L' \subseteq \Gamma^* \) be a regular language. Then so is \( h^{-1}(L') \).

To establish this result, we use the fact that \( L' \) is accepted by some DFA \( M' = (K', \Gamma, \delta', s', F') \). Let \( M \) denote the DFA that will accept \( h^{-1}(L') \), where \( M = (K, \Sigma, \delta, s, F) \) where \( K = K' \), \( F = F' \), and \( s = s' \). We define

\[
\delta(q, a) = \delta'(q, h(a)).
\]

We can establish by induction that

\[
\delta(q, x) = \delta'(q, h(x)), \text{ where } x \in \Sigma^*.
\]

Finally, we observe that \( x \in L(M) \iff \delta(s, x) \in F \iff \delta(s, h(x)) \in F \iff h(x) \in L(M') \iff x \in h^{-1}(L(M')) \).

Let \( L \) be a language over the one letter alphabet \( \{a\} \). Then \( L \) is regular if and only if the set of non-negative integers \( U = \{m : a^m \in L\} \) is ultimately periodic; that is there are integers \( n \geq 0 \) and \( p > 0 \) such that for all \( m \geq n \), \( m \in U \) if and only if \( m + p \in U \). We call the number \( p \) the period of \( U \).

To see this result, we use the fact that a regular language is accepted by a DFA.

Complexity of Algorithms for Finite Automata and Regular Languages:

(a) There is an exponential algorithm which constructs an equivalent deterministic finite automaton for a given non-deterministic finite automaton.

(b) There is an exponential algorithm that constructs an equivalent regular expression for a given non-deterministic automaton.

(c) There is a polynomial algorithm which constructs an equivalent non-deterministic finite automaton for a given regular expression.

(d) There is a polynomial algorithm that constructs a minimum state deterministic finite automaton for a given deterministic finite automaton.
(e) There is a polynomial algorithm which decides when two deterministic finite automata are equivalent.

(f) There is an exponential algorithm which decides when two non-deterministic finite automata are equivalent.

(g) For a given regular language \( L \) and string \( w \) there is an algorithm that decides if \( w \) belongs to \( L \) whose complexity is linear in the length of \( w \).

(h) For a given regular language \( L = L(M) \) where \( M \) is a deterministic finite automaton and string \( w \) there is an algorithm that decides if \( w \) belongs to \( L \) whose complexity is linear in the length of \( w \).

For a given regular language \( L = L(M) \) where \( M \) is a non-deterministic finite automaton and string \( w \) there is an algorithm that decides if \( w \) belongs to \( L \) whose complexity is \( O(|K|^2 |w|) \).