Last Lecture Covered...

- Regular expressions and languages, and some of their properties

This Lecture Covers Chapter 4 of HMU
Properties of Regular Languages

- Pumping Lemma for regular languages
- Some more properties of regular languages
- Decision properties of regular languages
- Equivalence and minimization of automata

Background Reading: Chapter 4 of HMU.
**Pumping Lemma**

- If a language is given by a regular expression, or a DFA, it is regular.

- What can we say if a language is defined by enumeration or by a predicate?

- Is \( L = \{ w \in \{0, 1\}^* : w \text{ does not contain } 10 \} \) regular?

- Is \( L = \{ 0^n 1^n : n \geq 0 \} \) regular?

  How do we answer such questions without delving into each case?

- Is there an inherent structure to the strings in a regular language?

---

**Theorem 1:** Let \( L \) be a regular language. There there exists an \( n \in \mathbb{N} \) (depending on \( L \)) such that for any string \( w \in L \) with \( |w| \geq n \), there exist strings \( x, y, z \) such that:

1. \( w = xyz \)
2. \( |xy| \leq n \)
3. \( |y| > 0 \)
4. \( xy^iz \in L \) for \( i \in \mathbb{N} \)
Pumping Lemma

Theorem 1: Let \( L \) be a regular language. There exists an \( n \in \mathbb{N} \) (depending on \( L \)) such that for any string \( w \in L \) with \( |w| \geq n \), there exist strings \( x, y, z \) s.t.: 

1. \( w = xyz \)  
2. \( |xy| \leq n \)  
3. \( |y| > 0 \)  
4. \( xy^iz \in L \) for \( i \geq 0 \)

Proof: Let DFA \( A = (Q, \Sigma, \delta, q_0, F) \) accept \( L \), and let \( n := |Q| \).

The claim is vacuously true if \( L \) contains only strings of length \( n - 1 \) or less.

Suppose \( L \) contains a string \( w = s_1 \cdots s_k \in L \) with \( |w| = k \geq n \).

Then, there must be a sequence of transitions that move \( A \) from \( q_0 \) to some final state upon reading \( w \).

\[
q_0 = q_{i_0} \xrightarrow{s_1} q_{i_1} \xrightarrow{s_2} q_{i_2} \cdots \xrightarrow{s_n} q_{i_n} \cdots \xrightarrow{s_k} q_{i_k} \in F
\]

\( n \) symbols and \( n + 1 \) states

\textbf{SOME} state must be visited (at least) twice. Let \( q_{i_a} = q_{i_b} \) for \( i_0 \leq i_a < i_b \leq i_n \).

\[
q_0 = q_{i_0} \xrightarrow{s_1} \cdots \xrightarrow{s_{i_a}} q_{i_a} \xrightarrow{s_{i_a+1}} \cdots \xrightarrow{s_{i_b}} q_{i_b} \xrightarrow{s_{ib+1}} \cdots \xrightarrow{s_n} q_{i_n} \cdots \xrightarrow{s_k} q_{i_k} \in F
\]

(4) holds since the path for \( xy^iz \) is derived from the above either by deleting the subpath between \( q_{i_a} \) and \( q_{i_b} \) or by repeating it. All such paths end in \( q_{i_k} \in F \).
Pumping Lemma: Applications

• $L = \{0^n1^n : n \geq 0\}$ is not regular.

  Suppose it is. Let DFA $A = (Q, \Sigma, \delta, q_0, F)$ accept $L$, and let $\ell := |Q|$

  Consider $0^\ell 1^\ell$.

  By the Pumping Lemma, there must exist $x, y, z$ s.t. $xyz = 0^\ell 1^\ell$ and $|y| > 0$

  $$y = 0^j \quad j > 0$$

  $\Downarrow$

  $$0^{\ell+j}1^\ell \in L$$

  $$y = 0^j1^k \quad (j, k > 0)$$

  $\Downarrow$

  $$0^\ell1^k0^j1^\ell \in L$$

  $$y = 1^k \quad (k > 0)$$

  $\Downarrow$

  $$0^\ell1^{\ell+k} \in L$$

  A contradiction in each case.

• $L = \{w \in \{0, 1\}^* : |w| \text{ is a prime}\}$ is not regular.

• $L = \{ww^R : w \in \{0, 1\}^*\}$ is not regular. \([w^R = w \text{ read from right to left}]\).
Some More Properties of Regular Languages

- We already know regular languages are closed under:
  union, intersection, concatenation, Kleene-\(^\ast\) closure, and difference.

- We’ll see three more operations under which regular languages are closed.

Let \( L^R \) be the language obtained by reversing each string \((01)^R = 10\)

**Theorem 2:** Let \( L \) be regular. Then \( L^R := \{w^R : w \in L\} \) is also regular.

**Proof:**

\( L \)

1. Reverse arrows;
2. Swap final and initial states; and
3. Introduce \( \epsilon \)-transitions to make initial state unique.

\( L^R \)
Some More Properties of Regular Languages

• A homomorphism is a map \( h : \Sigma_1 \rightarrow \Sigma_2^* \).

• The map can be extended to strings by defining \( s_1 \cdots s_k \overset{h}{\mapsto} h(s_1) \cdots h(s_k) \).

Theorem 3: Let \( L \) be regular. Then \( h(L) := \{ h(w) : w \in L \} \) is also regular.

Proof: Let \( E \) be the regular expression corresponding to \( L \).

Let \( h(E) \) be the expression obtained by replacing symbols \( s \in \Sigma_1 \) by \( h(s) \).

Then \( h(E) \) is a regular expression over \( \Sigma_2 \).

By a straightforward induction argument, we can show that \( L(h(E)) = h(L(E)) \).
Some More Properties of Regular Languages

• A homomorphism is a map $h : \Sigma_1 \rightarrow \Sigma_2^*$.
• The map can be extended to strings by defining $s_1 \cdots s_k \mapsto h(s_1) \cdots h(s_k)$.

**Theorem 4**: Let $L$ be regular. Then $h^{-1}(L) := \{w : h(w) \in L\}$ is also regular.

**Proof**: Let DFA $A = (Q, \Sigma_2, \delta, q_0, F)$ accept $L$
Let DFA $B = (Q, \Sigma_1, \gamma, q_0, F)$ where

$$\gamma(q, s) = \hat{\delta}(q, h(s))$$

[Depending on the input $B$ mimics none, one, or many transitions of $A$]

By definition, $\epsilon \in L(A)$ iff $q_0 \in F$ iff $\epsilon \in L(B)$
By induction, we can show that

$$s_1 \cdots s_k \in L(B) \iff h(s_1) \cdots h(s_k) \in L(A) = L$$

Hence, $B$ accepts $h^{-1}(L)$. 
Decision Properties of Regular Languages

- DFAs and regular expressions are **finite representations** of regular languages.
- How do we ascertain if a particular property is satisfied by a language?

Q1. Is the language accepted by a DFA is non-empty?
Q2. Does the language accepted by a DFA contain a given string \( w \)?
Q3. Is the language accepted by a DFA infinite?
Q4. Do two given DFAs accept the same language?
Q5. Given two DFAs \( A \) and \( B \), is \( L(A) \subseteq L(B) \)?

- We will look at these five assuming that languages are defined by DFAs. [If the language is specified by an expression, we can convert it to a DFA!]
Decision Properties

- **Emptiness:** If one is given a DFA with $n$ states that accepts $L$, we can find all the states reachable from the initial state in $O(n^2)$ time. If no final state is reachable, $L$ must be empty.

- **Membership:** If one is given a DFA with $n$ states that accepts $L$, given string $w$, we can simply identify the transitions corresponding to $w$ one symbol at a time. If the last state is an accepting state, then $w$ must be in the language. This takes no more than $O(|w|)$ time steps.

- **Infiniteness:** We can reduce the problem of infiniteness to finding a cycle in the directed graph (a.k.a. transition diagram) of the DFA.

  First, delete any node unreachable from the initial node ($O(n^2)$ complexity).

  Next, delete nodes that cannot reach any final node ($O(n^3)$ complexity).

  Use depth-first search (DFS) to find a cycle in the remaining graph ($O(n^2)$ complexity).
Decision Properties

- **Inclusion:** Given $A = (Q_A, \Sigma, \delta_A, q_{A0}, F_A)$ and $A = (Q_B, \Sigma, \delta_B, q_{B0}, F_B)$, how do we ascertain if $L(A) \subseteq L(B)$?

$$L(A) \subseteq L(B) \iff L(A) \cap L(B)^c = \emptyset$$

Run $A$ and $B$ in parallel.

$L(A) \cap L(B)^c$: Accept if resp. paths ends in $F_A$ and $F_B^c$.

Use **product** DFA: Construct $C = (Q_C, \Sigma, \delta_C, q_C, F_C)$ defined by

- $Q_C = Q_A \times Q_B$ [Cartesian product]
- $q_C = (q_{A0}, q_{B0})$
- $\delta_C((q, q'), s) = (\delta_A(q, s), \delta_B(q', s))$ [Both DFAs are simulated simultaneously]
- $F_C = (F_A \times F_B^c)$ [C accepts those strings in $L(A)$ but not in $L(B)$]

$$L(A) \subseteq L(B) \iff L(C) = \emptyset$$
Decision Properties

• **Equivalence:** Given $A = (Q_A, \Sigma, \delta_A, q_{A0}, F_A)$ and $A = (Q_B, \Sigma, \delta_B, q_{B0}, F_B)$, how do we ascertain if $L(A) = L(B)$?

\[
L(A) = L(B) \Leftrightarrow L(A) \cap L(B)^c = \emptyset \\
L(A)^c \cap L(B) = \emptyset
\]

💡 Run $A$ and $B$ in parallel.

- $L(A) \cap L(B)^c$: Accept if resp. paths ends in $F_A$ and $F_B^c$.
- $L(A)^c \cap L(B)$: Accept if resp. paths ends in $F_A^c$ and $F_B$.

Use **product** DFA: Construct $C = (Q_C, \Sigma, \delta_C, q_C, F_C)$ defined by

- $Q_C = Q_A \times Q_B$ [Cartesian product]
- $q_C = (q_{A0}, q_{B0})$
- $\delta_C((q, q'), s) = (\delta_A(q, s), \delta_B(q', s))$ [Both DFAs are simulated simultaneously]
- $F_C = (F_A \times F_B^c) \cup (F_A^c \times F_c)$ [C accepts those strings that are precisely in one of $L(A)$ or $L(B)$]

\[
L(A) = L(B) \Leftrightarrow L(C) = \emptyset
\]
• Given two DFAs, we know how to test if they accept the same language.

• Is there a unique minimal DFA for a given regular language?

• Given a DFA, can we reduce the number of states without altering the language it accepts?

Clearly, the two DFAs accept the same language and state C is unnecessary.
How do we remove ‘unnecessary’ states without altering the underlying language?
DFA State Minimization

- State minimization requires a notion of **equivalence** or **distinguishability**
  of states

  - Clearly, **distinguishability** of two states must be based on **finality**

  states $p$ and $q$ are **equivalent** (or **indistinguishable**)
  $\iff \delta(p, w) \in F$ whenever $\delta(q, w) \in F$
  for every $w \in \Sigma^*$.

  $p$ and $q$ are **distinguishable** $\iff$
  exactly one of $\delta(p, w)$ or $\delta(q, w)$ is a final state.
  for some $w \in \Sigma^*$,

**Table Filling** Algorithm identifies equivalent and distinguishable pairs of states.

(A1) Any final state is **distinguishable** from a non-final state (and vice versa)

(A2) If $\begin{cases} p \text{ and } q \text{ are distinguishable} \\ \delta(p', s) = p \\ \delta(q', s) = q \end{cases}$

then $p'$ and $q'$ must be distinguishable

- Apply A2 repeatedly until no new distinguishable pair is found.
**DFA State Minimization: An Example**

**Theorem 5:** Any two states without a $\times$ sign are equivalent.

**Proof idea:** If two states are distinguishable, the algorithm will fill a $\times$ eventually.
DFA State Minimization: An Example

- How do we use equivalence to minimize states?

1) Delete states not reachable from start states
2) Delete states that cannot reach any final state
3) Find distinguishable and equivalent pair of states
4) Find equivalence classes of indistinguishable states.
   In this example: \{A\}, \{B\}, \{C, E\}, \{D, F\}, \{G\}
5) Simply collapse each equivalence class of states to a state
6) Delete parallel transitions with same label.

Remark: The resultant transition diagram will be a DFA. [Why?]
Table-filling: Other Uses

- Test equivalence of languages accepted by 2 DFAs.

  Given $A = (Q_A, \Sigma, \delta_A, q_{A0}, F_A)$ and $B = (Q_B, \Sigma, \delta_B, q_{B0}, F_B)$:
  
  - Rename states in $Q_B$ so that $Q_A$ and $Q_B$ are disjoint.
  
  - View $A$ and $B$ together as one DFA
    (ignore the fact that there are 2 start states)
  
  - Run table-filling on $Q_A \cup Q_B$.

  $q_{A0}$ and $q_{B0}$ are indistinguishable $\Leftrightarrow L(A) = L(B)$.

  [Why?] *If* $w$ *distinguishes* $q_{A0}$ *from* $q_{B0}$ *then* $w$ *cannot be in both* $L(A)$ *and* $L(B)$

- Suppose a DFA $A$ cannot be minimized further by table-filling.
  Then, $A$ has the least number of states among all DFAs that accept $L(A)$