Finite Automata

Finite automata model systems with a *fixed* amount of memory, e.g, door controller

- Three components:
  - front pad, door, rear pad
  - door swings open to the rear
- Door is in two possible *states*: CLOSED or OPEN
- Four possible *signals* (or *inputs*): FRONT, REAR, BOTH, NEITHER
- When CLOSED: NEITHER or BOTH $\rightarrow$ CLOSED, FRONT $\rightarrow$ OPEN; REAR $\rightarrow$ CLOSED
- When OPEN: NEITHER $\rightarrow$ CLOSED; FRONT or REAR $\rightarrow$ OPEN; BOTH $\rightarrow$ OPEN

Finite Automata

- Door controller can be represented by a *state diagram*.

Other Examples

- *Lexical analyzer* of a compiler for detecting identifiers, keywords, and punctuation
- *Model checking techniques* for verifying finite state systems (e.g., communication protocols, hardware systesms)
- In the probabilistic setting, Markov chains (e.g. for PageRank)

Example of Finite Automata

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

Start state is q_1. *Accept state* is q_2.

Other examples:

- counting mod n
- $M = \{w \in \{0,1\}^* \mid w \text{ ends with a } 1\}$.
- complement

Formal Definition

A *finite automaton (FA)* is a structure $M = (Q, \Sigma, \delta, q_0, F)$, where

- Q is a finite set of *states*,
- Σ is the input symbols or alphabet,
- δ, the *transition function*, e.g., $\delta(q, a) = q'$ where $q, q' \in Q$ and $a \in \Sigma$
- $q_0 \in Q$ is the *start state*,
- F is a subset of Q; elements of F are called *accept states* or *final states*.
- The *language of* the machine M is the set L of all strings that M *accepts*. We write $L(M) = L$ and say M *accepts (or recognizes)* L. If the machine M accepts no strings then $L(M) = \emptyset$.

Formal Definition of Computation (Acceptance)

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton, and let $w = w_1 w_2 ... w_n$ be a string over Σ.

Then M *accepts* w if there is a sequence of states $r_0, ..., r_n$ in Q s.t.

1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}$
3. $r_n \in F$

M *accepts* or *recognizes* L if $L = L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$. A language is called a *regular language* if some finite automaton accepts it.

More Examples

Construct a finite automaton accepting the following language:

$$\{x \in \{a, b\}^* \mid x \text{ contains a substring of 3 consecutive } a\text{'s}\}$$

Another Example

Construct a finite automaton accepting the following language:

$$\{x \in \{a, b\}^* \mid x \text{ does not contain a substring of 3 consecutive } a\text{'s}\}$$

Operations on Languages

Recall:

- A *language* is set of strings over an alphabet.
- We may apply set operations like union, intersection, and set difference to languages.
- The *union* of L and M is denoted $L \cup M$, and is the set of strings that are in either L or M.
- The complement of a language L is $\Sigma^* - L = \{x \in \Sigma^* \mid x \notin L\}$, denoted $\bar{L}$ if Σ is understood.

Operations on Languages (cont'd)

- If L_1 and L_2 are languages over Σ their concatenation is

$$L_1 L_2 = L_1 \cdot L_2 = \{w \in \Sigma^* \mid w = xy \text{ for some } x \in L_1 \text{ and } y \in L_2\}.$$

- The *Kleene star* of a language L is the set of all strings obtained by concatenating zero or more strings from L. Thus,

$$L^* = \{w \in \Sigma^* \mid w = w_1 ... w_k \text{ for some } k \geq 0 \text{ and } w_1, ..., w_k \in L\}.$$

Closure Properties of Regular Languages

A set is *closed* under some operation if applying that operation to elements of the set returns in another element of the set.

Theorem: If L_1 and L_2 are regular languages then so is $L_1 \cup L_2$.

Proof Idea: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ be the automata that recognize L_1 and L_2, respectively. We construct $M = (Q, \Sigma, \delta, q_0, F)$ which recognizes $L_1 \cup L_2$ by *simulating* both of these machines *concurrently*, and accepting if one of them accepts.

Union Construction

1. $Q = \{(r_1, r_2) \mid r_1 \in Q_1, r_2 \in Q_2\}$

2. For each $(r_1, r_2) \in Q$ and each $a \in \Sigma$, $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$.

3. $q_0 = (q_1, q_2)$

4. $F = \{(r_1, r_2) \mid r_1 \in F_1 \textit{ or } r_2 \in F_2\}$.

Why this construction works: remembers the pairs of states the machine is in (“concurrent execution”.) Similarly, closed under intersection.

Closed Under Concatenation

Theorem: If L_1 and L_2 are regular languages, then so is $L_1 \cdot L_2$.

Product construction doesn't help here... need to do two strings consecutively, not concurrently.