Lecture Outline

Finite Automata

The *finite state machine* or *finite automaton* is the simplest computational model of limited memory computers.

Example: an automatic door opener at a supermarket decides when to open or close the door, depending on the input provided by its sensors.

Finite Automata

The *finite state machine* or *finite automaton* is the simplest computational model of limited memory computers.

Example: an automatic door opener at a supermarket decides when to open or close the door, depending on the input provided by its sensors.

Finite automata are designed to solve *decision problems*, i.e., to decide whether a given input satisfies certain conditions.

Examples of decision problems:

- does a given string have an even number of 1's;
- is the number of 0's (in a given string) multiple of 4;
- does a given string end in 00.

Example: Door Opener

states: closed, open

input conditions: front, rear, both, neither

nonloop transitions:

closed $\rightarrow$ open on front

open $\rightarrow$ closed on neither

Example: Door Opener

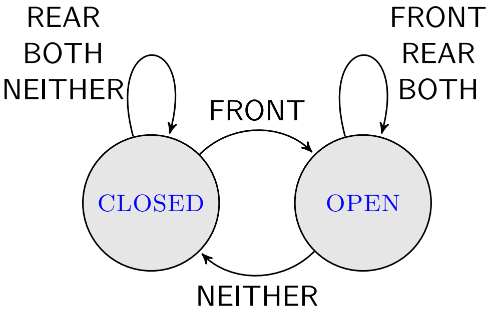
states: closed, open

input conditions: front, rear, both, neither

nonloop transitions:

closed $\rightarrow$ open on front

open $\rightarrow$ closed on neither



Formal Definition

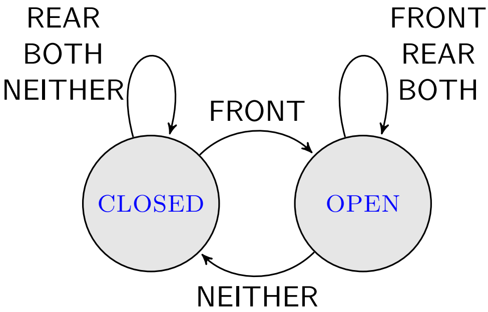
Definition

A *deterministic finite automaton (DFA)* is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

- Q is a finite set whose members are called *states*,
- Σ is a finite *alphabet* whose members are called *symbols*,
- $\delta : Q \times \Sigma \to Q$ is the *transition function*,
- $q_0 \in Q$ is the *start state*, and
- $F \subseteq Q$ is the set of *accept states* (or *final states*).

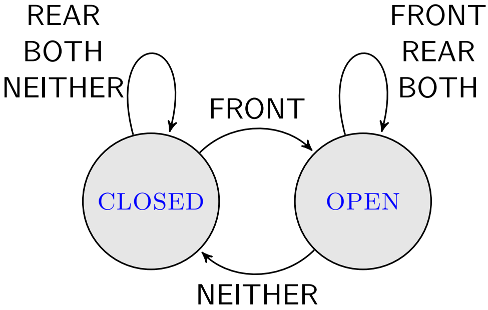
DFA computation can be described informally using a tape, cells with symbols, and a finite-state control with a read head advancing over the input. Given an input string over Σ (written on the input tape), an automaton reads its symbols one-by-one and changes its state (starting from q_0) according to δ. The automaton "accepts" the input if its resulting state (after reading of the input string is complete) belongs to F; otherwise "rejects".

What is what in Door Opener



Q: Is it a DFA?

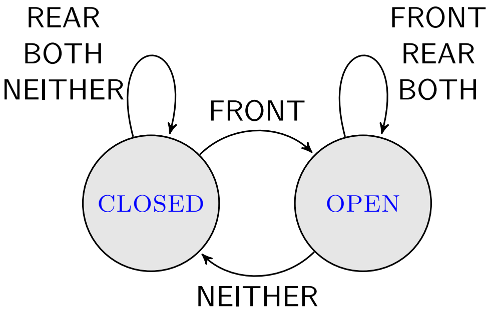
What is what in Door Opener



This automaton is not a DFA since it supposedly operates infinitely long and thus there are no starting state q_0 and final states F defined. However, it does correspond to some Q, Σ, δ.

Recall that Q is a finite set whose members are called *states*. What are the states of the Door Opener?

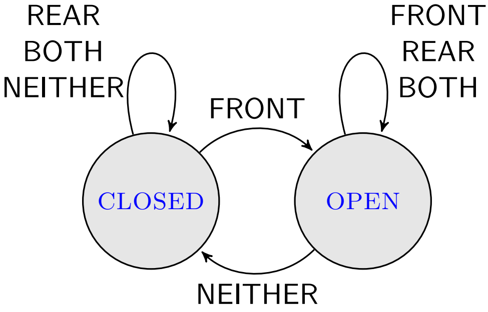
What is what in Door Opener



This automaton is not a DFA since it supposedly operates infinitely long and thus there are no starting state q_0 and final states F defined. However, it does correspond to some Q, Σ, δ.

$$Q = \{\mathrm{CLOSED}, \mathrm{OPEN}\}$$

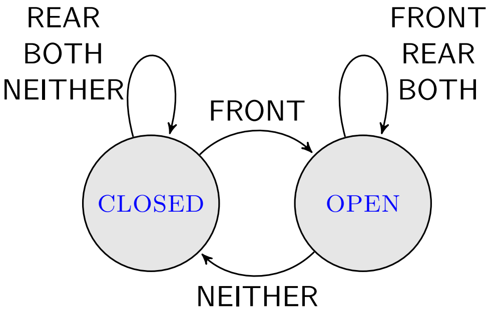
What is what in Door Opener



This automaton is not a DFA since it supposedly operates infinitely long and thus there are no starting state q_0 and final states F defined. However, it does correspond to some Q, Σ, δ.

Recall that Σ is a finite *alphabet* containing possible input *symbols*. What are they in the Door Opener?

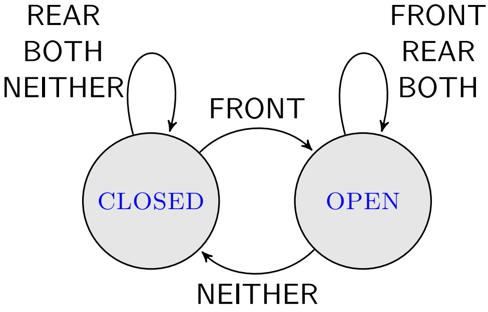
What is what in Door Opener



This automaton is not a DFA since it supposedly operates infinitely long and thus there are no starting state q_0 and final states F defined. However, it does correspond to some Q, Σ, δ.

$$\Sigma = \{\mathsf{FRONT}, \mathsf{REAR}, \mathsf{BOTH}, \mathsf{NEITHER}\}$$

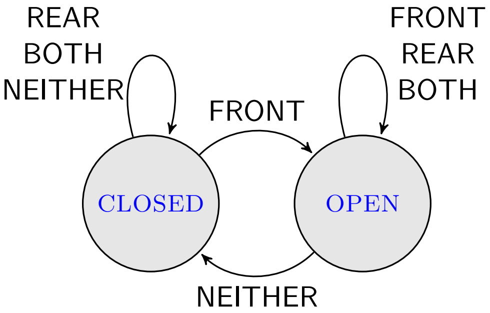
What is what in Door Opener



This automaton is not a DFA since it supposedly operates infinitely long and thus there are no starting state q_0 and final states F defined. However, it does correspond to some Q, Σ, δ.

Recall that transition function is a function $\delta : Q \times \Sigma \to Q$ defining how the automaton behaves.

What is what in Door Opener



This automaton is not a DFA since it supposedly operates infinitely long and thus there are no starting state q_0 and final states F defined. However, it does correspond to some Q, Σ, δ.

	FRONT	REAR	BOTH	NEITHER
CLOSED	OPEN	CLOSED	CLOSED	CLOSED
OPEN	OPEN	OPEN	OPEN	CLOSED

State Diagram of a DFA

- Nodes encode the states (elements of Q).
- Directed edges, labelled with symbols (elements of Σ), encode δ. Thus, each node has $|\Sigma|$ outgoing edges. Parallel edges can be combined into a single edge with multiple labels.
- An incoming edge from nowhere encodes the starting state.
- Nodes with double border encode accept states (elements of F).

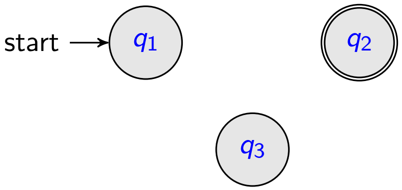
Drawing a State Diagram

Q: Draw a state diagram of a DFA M_1 with state set $Q = \{q_1, q_2, q_3\}$, alphabet $\Sigma = \{0, 1\}$, start state q_1, final state set $F = \{q_2\}$, and transition function δ given by the following table:

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_1

We start with drawing nodes (labeled with elements of Q),

marking the starting and accepting states:

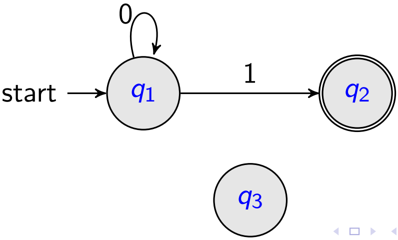


Drawing a State Diagram

Q: Draw a state diagram of a DFA M_1 with state set $Q = \{q_1, q_2, q_3\}$, alphabet $\Sigma = \{0, 1\}$, start state q_1, final state set $F = \{q_2\}$, and transition function δ given by the following table:

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_1

Then we draw outgoing edges (defined by δ) for the first node (q_1 is this example):

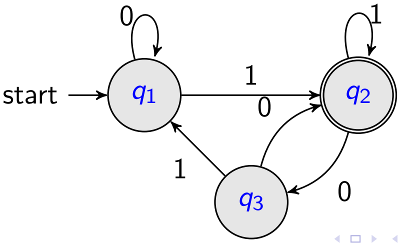


Drawing a State Diagram

Q: Draw a state diagram of a DFA M_1 with state set $Q = \{q_1, q_2, q_3\}$, alphabet $\Sigma = \{0, 1\}$, start state q_1, final state set $F = \{q_2\}$, and transition function δ given by the following table:

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_1

Similarly we draw outgoing edges for the other nodes and we are done:



Languages

Definition

Let Σ be an alphabet. We let Σ^* denote the set of all (finite) strings over Σ. A *language over* Σ is any subset of Σ^*, i.e., any set of strings over Σ.

We use languages to encode decision problems. Given an input (string), is the answer *"yes"* or *"no"*? Strings for which the answer is *"yes"* are called *yes-instances* and the others are *no-instances*. The language corresponding to a decision problem is the set of strings encoding yes-instances.

Recognizng DFA

Definition

Let M be a DFA with alphabet Σ and let $A \subseteq \Sigma^*$ be a language over Σ. We say that M *recognizes* A iff M accepts every string in A and rejects every string in $\overline{A} = \Sigma^* \setminus A$. We let $L(M)$ denote the language recognized by M.

Thus recognizing a language means being able to distinguish membership from nonmembership in the language, thus solving the corresponding decision problem. Every DFA recognizes a unique language (consisting of the strings it accept).

Designing a DFA

Q: Design a DFA recognizing the language $B = \{x \in \{0,1\}^\star \mid |x| \geq 2 \text{ and the 1st symbol of } x \text{ equals the last symbol of } x\}$.
In order to design such a DFA we need:

- to "record" the first symbol of the input (using states as the only available sort of memory);
- to distinguish whether the current symbol (that will be the last one at some point) matches the recorded first symbol;
- (at the end) accept if it does, reject otherwise.

DFA Computation

We now define a DFA computation formally.

Definition

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA and let $w \in \Sigma^*$ be a string over Σ. Suppose $w = w_1 w_2 \cdots w_n$ where $w_i \in \Sigma$ for all $1 \leq i \leq n$. The *computation of M on input w* is the unique sequence of states $(s_0, s_1, \ldots, s_n)$ where

- each $s_i \in Q$,
- $s_0 = q_0$, the start state, and
- $s_i = \delta(s_{i-1}, w_i)$ for all $1 \leq i \leq n$.

The computation $(s_0, \ldots, s_n)$ is *accepting* if $s_n \in F$, and is *rejecting* otherwise. If the former holds, we say that M *accepts* w, and if the latter holds, we say that M *rejects* w.

Thus $s_0, s_1, \ldots$ is the sequence of states that M goes through while reading w from left to right, starting with the start state.

Examples

Example: DFA that recognizes multiples of 3 in unary, binary.
Example: DFA that accepts strings over $\{a, b, c\}$ containing *abacab* as a substring. (Idea is useful for text search.)
Example: Strings over $\{0, 1\}$ with an even number of 1s. Strings over $\{0, 1\}$ with an even number of 1s or an odd number of 0s.

Definition

A language $A \subseteq \Sigma^*$ is *regular* iff some DFA recognizes it, i.e., $A = L(M)$ for some DFA M.

Regular Operations

Let A and B be two languages. We define the following regular operations:

Union: $A \cup B = \{x \mid x \in A \ \vee x \in B\}$

Concatenation: $A \circ B = \{xy \mid x \in A \ \wedge y \in B\}$

Star: $A^\star = \{x_1 x_2 \dots x_k \mid k \in \mathcal{Z} \wedge k \geq 0 \wedge \forall i = 1, 2, \dots, k, \ x_i \in A\}$

Theorem

If A and B are regular languages then so are $A \cup B$ and $A \circ B$. In other words, the class of regular languages is closed under the union and concatenation operations.

Read proofs in Sipser pp.44-47.