Finite Automata

CSCI 2210

2023-09-11 and 2023-09-13

State Machines

A **state machine** has a number of possible internal states that it can be in and a number of stimuli that it can receive.

When it receives a stimulus the machine may produce some response and may change to a different state.

Example: Vending Machine

Consider a vending machine with the following attributes:

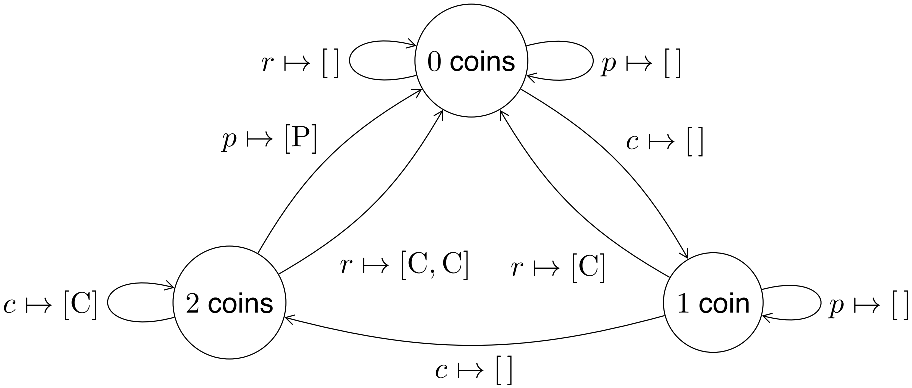
- sells only one kind of product (a can of soda),
- accepts only one kind of coin (a quarter),
- a soda costs two coins (50¢),
- has 3 stimuli: *insert coin* (c), *request product* (p), and *request refund* (r),
- has 2 responses: *release product* (P), and *release coin* (C).

subject to the following constraints:

- fulfills requests whenever possible,
- is “fair” (no cheating either way),
- balance can’t exceed price of product.

State Transition Graph

We can represent the behavior of the machine with a **state transition graph**:



Determinism

Our state transition graph is **deterministic**, in the sense that for each state and each possible stimulus there is exactly one sequence of responses and state transition.

This corresponds to a **function**:

$$\delta : \text{state} \times \text{stimulus} \to \text{responses} \times \text{state}.$$

	c	p	r
0	$([\,], 1)$	$([\,], 0)$	$([\,], 0)$
1	$([\,], 2)$	$([\,], 1)$	$([\mathrm{C}], 0)$
2	$([\mathrm{C}], 2)$	$([\mathrm{P}], 0)$	$([\mathrm{C}, \mathrm{C}], 0)$

Determining Language Membership

Let Σ be a nonempty finite set, which will think of as an **alphabet** of symbols.

Σ^* is the set of finite sequences of elements of Σ, which we think of as **words**.

A **language** over the alphabet Σ is a predicate on (or subset of) Σ^*.

One kind of computational task is, given a language $\mathrm{L} \subseteq \Sigma^*$ and a word $w \in \Sigma^*$, determine whether or not $w \in \mathrm{L}$.

Our first model of computation answers this question for a certain class of languages.

Deterministic Finite Automata

A **deterministic finite automaton** ("DFA"; synonym: "deterministic finite state machine") M has the following components:

input alphabet: a nonempty finite set of symbols Σ,

state set: a nonempty finite set of states Q,

start state: a chosen state $q_0 \in \mathrm{Q}$,

accept state set: a chosen subset of states $\mathrm{F} \subseteq \mathrm{Q}$,

state transition function: a function $\delta : \mathrm{Q} \times \Sigma \to \mathrm{Q}$.

DFA State Transition Graph

We can represent a DFA M using a state transition graph.

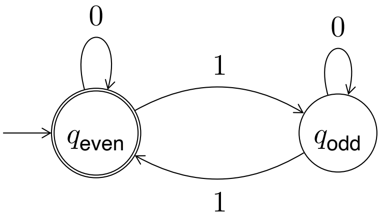
The vertex set is Q.

For each ordered pair $(q\ ,\ s) \in \mathrm{Q} \times \Sigma$ there is an edge $e : q \to \delta(q\ ,\ s)$ labeled by s.

In a diagram for this graph we annotate the vertex q_0 and those of F.

Example: DFA State Transition Graph Diagram

Let M_{par} be the DFA specified by:



This means:

- $\Sigma = \{0, 1\}$,
- $\mathrm{Q} = \{q_{\text{even}}, q_{\text{odd}}\}$,
- $q_0 = q_{\text{even}}$,
- $\mathrm{F} = \{q_{\text{even}}\}$,
- $\delta =$

	0	1
q_{even}	q_{even}	q_{odd}
q_{odd}	q_{odd}	q_{even}

Language of a DFA: Operational Semantics

Each DFA M over alphabet Σ determines a **language** $\mathrm{L}(\mathrm{M})$ according to the following **operational semantics**.

Given an input string $w = [w_0, w_1, \cdots, w_n] \in \Sigma^*$:

- M begins in the start state q_0,
- M inspects the symbols of w left-to-right one at a time,
- If M is in state q inspecting symbol w_i then M transitions to state $\delta(q\ , w_i)$,
- Upon exhausting w, M is in some state q_f,
- We say $w \in \mathrm{L}(\mathrm{M})$ just in case $q_f \in \mathrm{F}$.

Language of a DFA: State Transition Graph Interpretation

For a DFA M, each word $w \in \Sigma^*$ determines a path $p : q_0 \to q_f$ in the state transition graph $\mathrm{G}(\mathrm{M})$.

The state transition function δ determines where this path goes at each step.

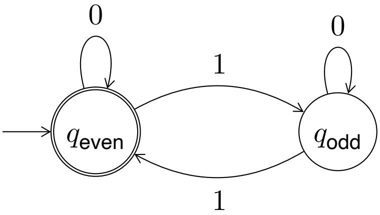
Since distinct edges share the same labels we write paths in state transition graphs with explicit vertices:

$$[(\mathrm{V}_0), e_0, (\mathrm{V}_1), e_1, (\mathrm{V}_2), e_2, (\mathrm{V}_3)]$$

$$\mathrm{V}_0 \xrightarrow{e_0} \mathrm{V}_1 \xrightarrow{e_1} \mathrm{V}_2 \xrightarrow{e_2} \mathrm{V}_3$$

Example: Language of M_{par}

$\mathrm{M}_{par} \quad := $



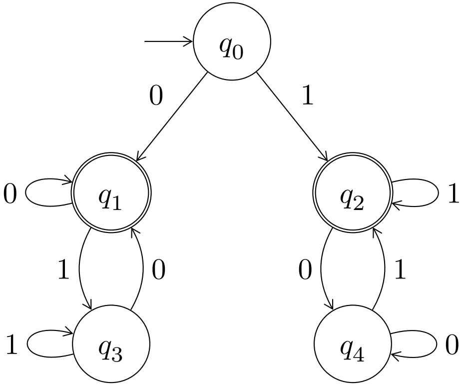
determines the language of even parity: $w \in \mathrm{L}(\mathrm{M}_{par})$ just in case w contains an even number of 1s.

for example:

- word 1001 determines path $[(q_{\text{even}}), 1, (q_{\text{odd}}), 0, (q_{\text{odd}}), 0, (q_{\text{odd}}), 1, (q_{\text{even}})]$
- word 010 determines path $[(q_{\text{even}}), 0, (q_{\text{even}}), 1, (q_{\text{odd}}), 0, (q_{\text{odd}})]$

Activity: Language of a DFA

Verify that the following state transition graph represents a DFA over alphabet $\Sigma := \{0, 1\}$ and give a description of its language.



Activity: DFA for a Language

Give a DFA for the language over alphabet $\Sigma := \{0, 1\}$ of words that begin with a 0.

Regular Languages

A language is called **regular** if there is some DFA that decides it.

To show that a language is regular, it suffices to produce the corresponding DFA.

Regular Language Constructions

An **empty language** contains no words ($L_\emptyset = \{\ \}$).

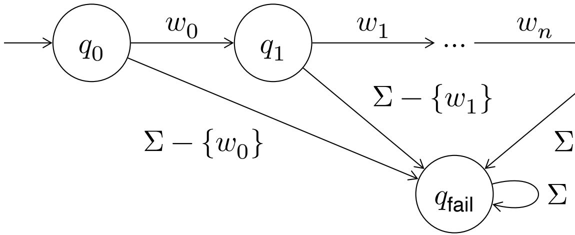
The empty language over the alphabet $\Sigma := \{0, 1\}$ is regular.

Regular Language Constructions

A **singleton language** contains exactly one word ($\mathrm{L}_w = \{w\}$).

Any singleton language over the alphabet $\Sigma := \{0, 1\}$ is regular.

Idea: construct a DFA with a "happy path" labeled by w from the start state to the only accept state, and add a captive "failure state" for falling off the happy path.



Regular Language Constructions

Given a language L over an alphabet Σ, its **complement language** $\bar{\mathrm{L}}$ contains exactly the words of Σ that are not contained in L:

$$\bar{\mathrm{L}} := \Sigma^* - \mathrm{L}.$$

The complement of a regular language over the alphabet $\Sigma := \{0, 1\}$ is regular.

Idea: swap the roles of the accepting and non-accepting states:

$$\mathrm{F}_{\bar{\mathrm{L}}} := \mathrm{Q} - \mathrm{F}_{\mathrm{L}}$$

Regular Language Constructions

Given languages L_0 and L_1 over an alphabet Σ, the **union language** $\mathrm{L}_0 \cup \mathrm{L}_1$ contains exactly the words of Σ^* that are contained in at least one of L_0 and L_1:

$$\overbrace{\mathrm{L}_0 \cup \mathrm{L}_1}^{\text{as languages}} := \underbrace{\mathrm{L}_0 \cup \mathrm{L}_1}_{\text{as subsets of } \Sigma^*}.$$

It seems like the union of regular languages should be regular.

Intuition: if DFA M_0 decides language L_0 and DFA M_1 decides language L_1, just run them both on string w and accept if either M_0 or M_1 accepts w.

Problem: two DFAs plus some control logic are not a DFA! We need another plan.

DFA for Union Language

Let DFA M_i be described by $(\Sigma, \mathrm{Q}_i, q_{0_i}, \mathrm{F}_i, \delta_i)$ for $i \in \{0, 1\}$.

We can make a DFA M for $\mathrm{L}_0 \cup \mathrm{L}_1$ with:

input alphabet: Σ

state set: $\mathrm{Q}_0 \times \mathrm{Q}_1$

start state: (q_{0_0}, q_{0_1})

accept state set: $\underbrace{(\mathrm{F}_0 \times \mathrm{Q}_1)}_{\text{accepted by } \mathrm{M}_0} \cup \underbrace{(\mathrm{Q}_0 \times \mathrm{F}_1)}_{\text{accepted by } \mathrm{M}_1}$

state transition function: $\delta : \mathrm{Q}_0 \times \mathrm{Q}_1 \times \Sigma \to \mathrm{Q}_0 \times \mathrm{Q}_1$ given by:

$$(q_x, q_y, s) \mapsto (\delta_0(q_x, s), \delta_1(q_y, s))$$