Tutorial 1

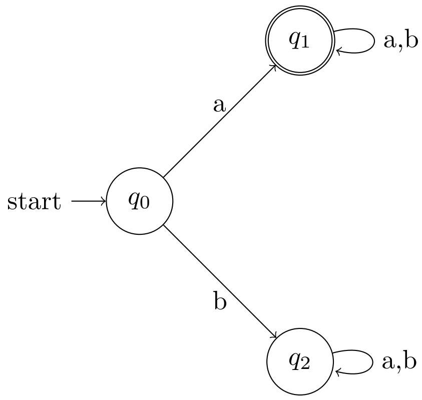
Gidon Rosalki

2025-03-26

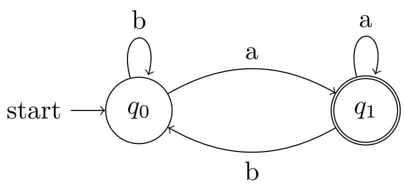
Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_computability_comple`
Given a python program that sorts lists, can you write another program that verifies whether or not the sorter will always return correctly sorted lists? In fact we can write a program such as this.
Given a map, can you colour in each country in one of 3 colours, such that no 2 countries of the same colour share a border? This is an open question, and solving it will get you 100 on the course. Note, this is also proving that P=NP, and will also get you $\$1,000,000$

1 Deterministic Finite Automata

Consider the following DFA:



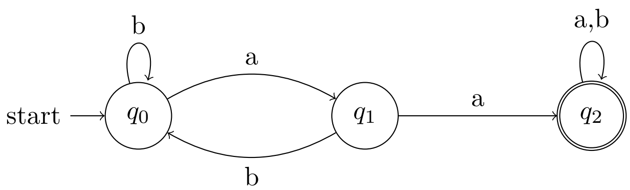
If we run this on abb, we start by moving to q_1, and staying there, and therefore the word is accepted. If we try on b, or ε then neither is accepted. Let's try another machine:



Here we can see that aba is accepted. In fact, this automaton accepts all words that finish with a.
Let us make an automaton that accepts all words that contain aa. We want it to remember if aa has ever appeared, and shall achieve this by detecting an a, and then another a as follows:

- q_0 is the words that do not contain aa, and do not finish in a
- q_1 is the words that do not contain aa, and finish in a
- q_2 is the words that contain aa

In order to prove the correctness of the automaton, we will need to prove the above theorems. To prove this we induct on word length.



2 Definitions

2.1 Languages

Definition 2.1 (Σ **- alphabet**)**.** *The **alphabet**, written as Σ is a finite non empty set. Its elements are called **letters**.*

$\Sigma = \{a, b\}$, then for all $n \in \mathbb{N}$,

$$\Sigma^n := \{(\sigma_1, \ldots, \sigma_n) : \sigma_1, \ldots, \sigma_n \in \Sigma\}$$

and

$$\Sigma^0 = \{\varepsilon\}$$

the empty sequence.

Definition 2.2 (Σ^*)**.**

$$\Sigma^* := \bigcup_{n=0}^{\infty} \Sigma^n$$

Definition 2.3 (Language)**.** *A language L over the alphabet Σ is $L \subseteq \Sigma^*$, also known as a set of words.*

So, given those definitions,

- $L_1 = \{ab, a, \varepsilon, bbb\}$ is a finite language
- $L_2 = \{w \in \Sigma^* : w \text{ starts with } a\}$ is an infinite language
- $L_3 = \{w \in \Sigma^* : |w| < 24\}$ is a finite language

2.2 DFA

Definition 2.4 (DFA)**.** *The DFA A is a vector of 5 things: $A = (\Sigma, Q, q_0, F, \delta)$ where*

- *Σ is an alphabet*
- *Q is the non empty finite set of states*
- *$q_0 \in Q$ is the starting state*
- *$F \subseteq Q$ is the set of accepted finishing states*
- *δ is the transition function $\delta : Q \times \Sigma \to Q$*

Definition 2.5 (Running a DFA on a word)**.** *Given $w = w_1 \ldots w_n \in \Sigma^*$, a running of A on w is $r_1, r_1, \ldots, r_n \in Q$ such that*

- $r_0 = q_0$
- $\forall 0 \leq i < n,\ r_{i+1} = \delta(r_i, w_{i+1})$

Definition 2.6 (Acceptance)**.** *We will say that the DFA A **accepts** w **if and only if** $r_n \in F$*

Definition 2.7 (DFA language)**.** *The language of the DFA is the set of accepted words:*

$$L(A) = \{w \in \Sigma^* : A \text{ accepts } w\}$$

For the first DFA example we did, we may formally define it as follows:

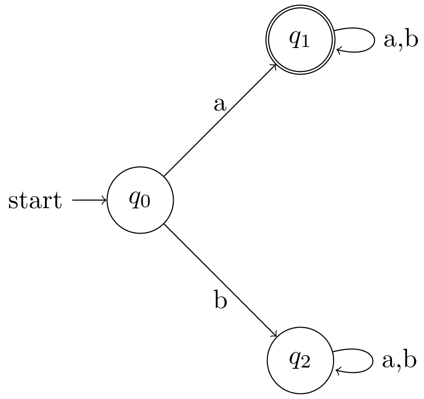
- $\Sigma = \{a, b\}$
- $Q = \{q_0, q_1\}$
- $F = \{q_1\}$
- The initial state is q_0
- δ is

	a	b
q_0	q_1	q_2
q_1	q_1	q_1
q_2	q_2	q_2

Table 1: δ

3 Formally proving what is the language of a DFA

Consider the DFA



In the words of the formal tutorial, we need to know how to prove its language, and an DFA's language, at gunpoint. I suspect that this is a threat, and that the midterm exam is going to be... Intense.

Theorem 1.

$$L(A) = L$$

Proof . We will want to prove that the words that finish their runs at each situation are as follows x are:

1. q_0 - The empty word
2. q_1 - Words that start with a
3. q_2 - Words that do not start with a, and are not the empty word

This is sufficient since

$$\begin{aligned} w \in L &\Leftrightarrow w \text{ starts with } a \\ &\Leftrightarrow \text{The running of } w \text{ finishes at } q_1 \\ &\Leftrightarrow A \text{ accepts } w \\ &\Leftrightarrow w \in L(A) \end{aligned}$$

We will prove by induction on the length of w:
Basis: $|w| = 0 \implies w = \varepsilon \implies$ the final state is q_0, as required
Inductive hypothesis: Let there be $w : |w| = n$, then the above requirements hold.
Inductive step: Let there be $w : |w| = n + 1,\ w = w'\sigma,\ |w'| = n, \sigma \in \Sigma$. We will split into situations, according to the state in which A finishes w'

1. $q_0 \implies w' = \varepsilon \implies w = \sigma$. We will split into situations by σ. If $\sigma = a \implies w = a$ and so we want the run to finish at q_1, and indeed from the definition $\delta(q_0, a) = q_1$.
   If $\sigma = b$, then similarly to $\sigma = a$
2. $q_1 \implies w'$ starts with a (from the induction hypothesis). Therefore, from the definition, $\forall \sigma \in \Sigma,\ \delta(q_1, \sigma) = q_1$
3. q_2 exactly like the previous.

□

4 The extended transition function

We write the extended transition function as δ^*. We defined earlier that $\delta : Q \times \Sigma \to Q$, and we will similarly define $\delta^* : Q \times \Sigma^* \to Q$ where

$$\forall q \in Q,\ w \in \Sigma^*,\ \delta^*(q, w) = \begin{cases} q, & \text{if } w = \varepsilon \\ \delta(\delta^*(q, w'), \sigma), & \text{if } w = w'\sigma \end{cases}$$

Theorem 2. *For all $q \in Q$ and $w, w' \in \Sigma^*$, it is true that*

$$\delta^* (q, w \cdot w') = \delta^* (\delta^* (q, w), w')$$

Proof . By induction on $|w'|$:
Basis: $|w'| = 0$:

$$\begin{aligned}\delta^* (q, w \cdot w') &= \delta^* (q, w \cdot \varepsilon) \\ &= \delta^* (q, w) \\ &= \delta (\delta^* (q, w), \varepsilon) \\ &= \delta^* (\delta^* (q, w), w')\end{aligned}$$

Step: We will assume for $|w| = n$, and prove for $|w'| = n + 1$. Note that $w' = w''\sigma$, $|w''| = n, \sigma \in \Sigma$:

$$\begin{aligned}\delta^* (q, w \cdot w') &= \delta^* (q, w \cdot w'' \cdot \sigma) \\ \text{Definition of } \delta^* &= \delta (\delta^* (qw \cdot w''), \sigma) \\ \text{Inductive hypothesis} &= \delta (\delta^* (\delta^* (q, w), w''), \sigma) \\ \text{Definition of } \delta^* &= \delta^* (\delta^* (q, w), w''\sigma) \\ &= \delta^* (\delta^* (q, w), w')\end{aligned}$$

□

5 Regularity of Leven

5.1 Regular languages

Definition 5.1. *L is a regular language if there exists a DFA A that determines it. The collection of regular languages is called REG*

$$REG \stackrel{def}{=} \{L \subseteq \Sigma^* : \exists A : L(A) = L\}$$

We will define $L_{EVEN} = \{w \in L : |w| \mod 2 = 0\}$. Is L_{EVEN} also regular? Yes, in fact it is.

5.2 Intuition

L is regular $\implies$ there exists $A = (Q, \Sigma, \delta, q_0, F)$ such that $L(A) = L$. We want to build an automaton $A' = (Q', \Sigma, \delta', q'_0, F')$ such that $L(A) = L_{EVEN}$

5.3 Proof (sketch)

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA that determines L. In order to construct a DFA for L_{EVEN}, we can define a new automaton that tracks both the state of A, and tracks if the number of bits in the input string so far is positive or negative. We can do this by defining $A' = (Q', \Sigma, \delta', (q_0, 0), F')$ where

- $Q' = Q \times \{0, 1\}$, where the second number tracks if the number of input bits so far is odd or even
- $\forall q \in Q,\ p \in \{0, 1\},\ a \in Sigma,\ \delta'((q, p), a) = (\delta(q, a), 1 - p)$
- The initial state is $(q_0, 0)$
- $F' = \{(q, 0) : q \in F\}$, which is all the accepting states of the original automaton, but ensuring that they have an even number of bits.

Since A' is a DFA, then the language it accepts L_{EVEN} is regular.