Foundations of Computer Science

Leonardo Biason

October 28, 2024

Chapter 1

Languages

While we may think that computers are just machines, there is an incredible work behind that aims to make some kind of language that computers can understand. It is well renewed that computers act depending on streams of binary code, but that same stream of binary code can be considered as a language, something that the computers "speak" and "think" with. Let's define more in detail this "computers' language".

DEFINITION

Alphabeth and Strings

An **alphabet** Σ is a **non-empty**, **finite set**, which contains elements called **symbols** (or **characters**). For instance, the following two sets are considered alphabets:

- $\Sigma = \{0, 1\}$
- $\Sigma = \{a, b, c, \ldots, x, y, z\}$

A **string** w over an alphabet Σ is a **sequence of symbols**, all belonging to Σ, which are all written one after the other and aren't separated by other symbols

Strings also have different properties, such as a **length**, a **reverse** and the possibility to include one or more **substrings**:

- **Length**: defined as $|w|$, it denotes the **number of symbols** contained within w. If a string has length 0, then such string is called **empty string**, and is denoted with ϵ;
- **Reverse**: defined as w^R, the reverse is a string which contains all the symbols of w in the reverse order;
- **Substring**: we say that a string z is a substring of w if z appears consecutively within w.

EXAMPLE

1.0.0

For instance, the string $w = 00101$ has a length of 5, its reverse is $w^R = 10100$, and has, as possible substrings the following: $001, 10, \epsilon$. Its alphabet is $\Sigma = \{0, 1\}$

When we have two strings x and y, we can **concatenate** them by appending y at the end of x. The result is denoted as xy. Clearly, the length of the concatenated string is equal to the sum of the length of the two strings ($|xy| = |x| + |y|$). A specific notation x^k denotes the concatenation of string x with itself for k times. So, if x was for instance "01", then we would have that

$$x^0 = \epsilon \qquad x^1 = 01 \qquad x^2 = 0101 \qquad x^3 = 010101 \quad \text{and so on...}$$

By combining the definitions of concatenation and substring, we can define properly what a **prefix** and **suffix** is:

- **Prefix**: we say that a string x is a prefix of a string z if there exists a string y such that $z = xy$. Moreover, we say that x is a **proper prefix** of z if, additionally, $x \neq z$, so if $y \neq \epsilon$;
- **Suffix**: we say that a string x is a suffix of a string z if there exists a string y such that $z = yx$. Moreover, we say that x is a **proper suffix** of z if, additionally, $x \neq z$, so if $y \neq \epsilon$.

Given these tools, we can now define what a language is:

Definition

Language

A **language** is a **set of strings**. It is also defined as **prefix-free** if no member is a proper prefix of any other member

Languages can follow an order. There are different types of orders, one of which is the **lexicographic order**, which is defined by the order of the alphabet; we can think of it as the familiar dictionary order. Another order is the **shortlex order** (or **string order**), which is the same of the lexicographic order with the exception that shorter strings precede longer strings.

1.1 Deterministic Finite Automata (DFA)

We know that sometimes circuits can use the output from a given combination of inputs as input for the next output. In that case, we talk about circuits with states, or more specifically, we talk about **automata**.

Definition

Deterministic Finite Automaton (DFA)

We define a **finite automaton** as a tuple of 5 elements (Q, Σ, δ, q_0, F), where:

- Q is a finite set, denoting the **states** that the automaton can reach;
- Σ is a finite set, denoting the **alphabet** of the automaton;
- δ is a function, called **transition function**, which given Q and Σ returns the set of next states Q';
$$\delta : Q \times \Sigma \longmapsto Q'$$
- $q_0 \in Q$ is a state, and denotes the **starting state** of the automaton;
- $F \subseteq Q$ is a finite set, denoting the set of **accepted states** (or **final states**). An accepted state tells the automata if a string can be accepted or not.

Let's make an example:

1.1.1

Example

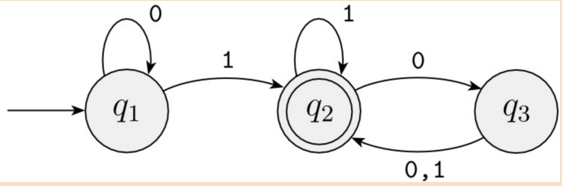
Consider the following tuple:

$$(\{q_1, q_2, q_3\}, \{0, 1\}, \delta, q_1, q_2)$$

It denotes that we have 3 possible states $\{q_1, q_2, q_3\}$, the alphabet is made of 2 symbols, $\{0, 1\}$, the starting state is q_1, the final state is q_2 and the transition function is δ. Such transition function could be the following:

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

We can represent it visually as follows:



Automata change states depending on the string that they receive in input. So when a string x is given to the automaton, it goes through the string symbol by symbol, and changes states depending on the input characters read and the result of the transition function. If, after reading the whole string, the automaton arrives at a final input, then we say that it **accepts** the string, otherwise it **rejects** it.

Language of an automaton and Regular Language

Definition

The **language of an automaton** M, denoted as $L(M)$, is the **set** of all those **input strings** which make the automaton **go** from the starting state **to** a **final state**.

More specifically: given an automaton $M = (Q, \Sigma, \delta, q_0, F)$ and let w be a string of n symbols where w_i denotes the i^{th} symbol of the string; let also that

$$\forall\, w_i \in w, \text{ then } w_i \in \Sigma$$

Then, we say that M accepts w if there exists a set of states $r = \{r_0, r_1, \dots, r_n\} \in Q$ which respects all the following three conditions:

- $r_0 = q_0$: the first state of r must coincide with the initial state;
- $\delta(r_i, w_{i+1}) = r_{i+1}, \forall i \in [0, 1, \dots, n-1]$: each character must lead to a transition of state;
- $r_n \in F$: the final state in r should be in F.

DEFINITION

M recognizes a language A If

$$A = \{w \mid M \text{ accepts } w\}$$

A **language** L is said to be **regular** if there exists a finite automaton M which accepts L

Note that it's possible for an automaton to have that $q_0 \in F$.

1.2 Non-Deterministic Finite Automata (NFA)

With DFA, we always knew that, given a current state q_i and a character $w_i \in \Sigma$, then we would **always know** what the **next state** would be. There is another type of automata which breaks this deterministic rule, and is called **non-deterministic finite automata**.

Non-Deterministic Finite Automaton (NFA)

DEFINITION

A **non-deterministic finite automaton** (**NFA**) is an automaton denoted by a tuple of 5 elements $(Q, \Sigma, \delta, q_0, F)$ where:

- Q is a finite set of **states** that the automaton can reach;
- Σ denotes a finite **alphabet** recognized by the automaton;
- δ is the **transition function**, which given Q and Σ_ϵ returns the **power set** of Q (denoted by $\mathcal{P}(Q)$);
  $$\delta : Q \times \Sigma_\epsilon \longmapsto \mathcal{P}(Q)$$
- $q_0 \in Q$ is the **starting state** of the automaton;
- $F \subseteq Q$ is the set of **accepted states**.

The power set $\mathcal{P}(Q)$ of a set Q is the set of all subsets of Q, for any Q; the alphabet denoted with Σ_ϵ is equal to $\Sigma \cup \{\epsilon\}$.

In practice, how do NFAs work? The idea of NFAs is that, at **any moment in time**, **several choices may exist** for deciding the next state, and also ϵ can be a choice. An NFA runs through an input in the following way:

1) Like for DFAs, we start from the initial state q_0;

2) Let q denote the current state and a the next symbol that the NFA receives as input, then:

    a) For every a-transition, the machine **splits** into a **different copy** and follows the path given by a, in parallel with all the other possible copies that might have been created. For instance, if from a state we have three outgoing transitions $0 \to q_i$, $1 \to q_j$ and $1 \to q_k$, then the machine will split into three different copies and, in parallel, will proceed on the three states, and the procedure will go on for each state;

b) For every ϵ-transition, the machine will split itself into a new copy **without reading the input**, and then each copy will proceed from the destination state given by each ϵ-transition;

c) If a state q has no outgoing transitions, and $q \notin F$, then that copy of the machine dies, alongside all its computations.

3) If **any of the copies** of the NFA arrives to a final state, then we say that the **NFA accepts** the input string.

We can think of an NFA as a machine that runs sort of "*parallel*" computations on independent threads, concurrently. It's also possible to think of an NFA as a **tree**, where the root of the tree is the root of the computation, and where each branch represents a possible choice that the machine has. If at least one of the leafs of the tree is a final state, then the machine accepts the input. Let's better formalize the acceptance of an NFA:

DEFINITION

Acceptance of an NFA

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and w a string over the alphabet Σ. We say that N **accepts** w if w can be rewritten as $w = y_1 y_2 ... y_m$ where $\forall i \in m,\ y_i \in \Sigma_\epsilon$ and there exists a sequence of states $r_0, r_1, ..., r_m$ in Q such taht the following three conditions are respected:

- $r_0 = q_0$ the initial states must coincide;
- $r_{i+1} \in \delta(r_i, y_{i+1})$: the next state must be in the set of states returned by the transition function given a specific state r_i and the next input character y_{i+1};
- $r_m \in F$: the final state must belong to F.

It is possible that $m \geq |w|$, because some $w_i \in w$ can be ϵ.

The possibility of using ϵ as a valid character is very helpful when it comes to recognizing one pattern of characters among a set of patterns. In the following example is better explained how to use this approach:

EXAMPLE

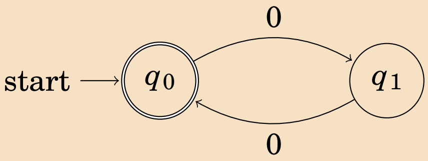
1.2.2

Suppose that we want to build an automaton (specifically, an NFA) which recognizes the words in the following language:

$$\left\{ 0^k \ : k \text{ is a multiple of 2 or 3} \right\}$$

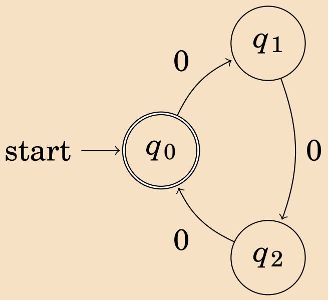
How can we build such automaton? Let's break it down into two smaller parts:

- if k must be a multiple of 2, then the automaton is as follows:

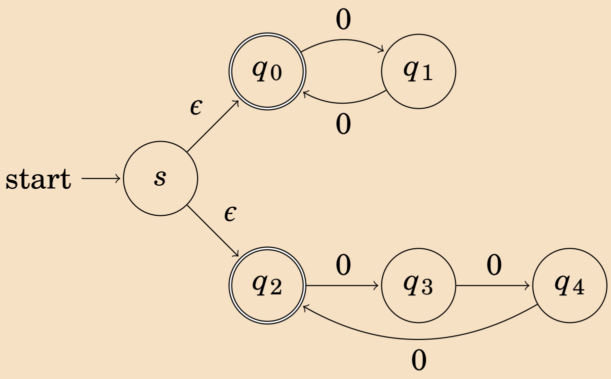


Example

- if k must be a multiple of 3, then the automaton is as follows:



We can merge these two automatons thanks to the ϵ-transition. This way the automaton will create a copy of itself for both cases (so that w has a number of zeroes which is either a multiple of 2 or 3, in the best case the number will be simultaneously a multiple of both 2 and 3), and if at least one of the two cases will end up in a final state, then the NFA will accept such string. Here is the final version:



Intuitively, we may think that a DFA can be expressed with an NFA, since DFAs are a specific version of NFAs where there are no ϵ-transitions and where, from each state, given a specific input character, there is only one possible action to perform. But can we say the same of the other way around? So, is there, for each NFA, the possibility to express is with a DFA? There is a theorem which states that it's possible, and we'll also see its proof:

Equivalence of DFAs and NFAs

Theorem

For every NFA N there exists a DFA M such that the language accepted by N is also accepted by M (so $L(N) = L(M)$)

Proof

Let N be an NFA such that $N = \{Q, \Sigma, \delta, q_0, F\}$ and let $A = L(N)$ denote the language accepted by N. Our objective is to devise a DFA $M = \{Q', \Sigma, \delta', q'_0, F'\}$ which accepts that same language A. Let's first consider a case where the NFA has no ϵ-transitions:

- $Q' = \mathcal{P}(Q)$: the set of states Q' would be equal to the power set of Q, so the set of all subsets that can be done with the states in Q;
- $q'_0 = \{q_0\}$: the starting state of N is equal to the starting state of M;
- $F' = \big\{R \in Q' \,|\, R \text{ contains an accepting state of } N\big\}$: the set of final states of M is equal to a subset R of Q', such that at least an accepting state of N is present in R.