

# Theory of Computation

Paolo Bettelini

## Contents

|          |  |          |
|----------|--|----------|
| <b>1</b> | <b>Fields of Study</b>                   | <b>2</b> |
| 1.1      | Complexity Theory . . . . .              | 2        |
| 1.2      | Computability Theory . . . . .           | 2        |
| 1.3      | Automata Theory . . . . .                | 2        |
| <b>2</b> | <b>Alphabets and Languages</b>           | <b>2</b> |
| <b>3</b> | <b>Deterministic Finite Automaton</b>    | <b>2</b> |
| <b>4</b> | <b>Regular Operations</b>                | <b>3</b> |
| 4.1      | Union . . . . .                          | 3        |
| 4.2      | Concatenation . . . . .                  | 3        |
| 4.3      | Kleene star operator . . . . .           | 3        |
| 4.4      | Regular language . . . . .               | 3        |
| 4.4.1    | Closure under union . . . . .            | 3        |
| <b>5</b> | <b>Nondeterministic Finite Automaton</b> | <b>4</b> |
| <b>6</b> | <b>Equivalence of DFAs and NFAs</b>      | <b>4</b> |
| 6.1      | DFA to NFA conversion . . . . .          | 4        |

# 1 Fields of Study

## 1.1 Complexity Theory

Classify problems according to their degree of "difficulty".

## 1.2 Computability Theory

Classify problems as being solvable or unsolvable.

## 1.3 Automata Theory

Compare different computation models.

# 2 Alphabets and Languages

An *alphabet* is a finite set of *symbols*. For example:  $\{a, b, c, \dots, z\}$

The set  $\{0, 1\}$  is the binary set. The empty string is denoted  $\lambda$ .

Note that  $\lambda \neq \emptyset \neq \{\lambda\}$ .

The length of a string  $w$  is denoted as  $|w|$ .

If  $\Sigma$  is an alphabet,

$$\Sigma_\lambda = \lambda \cup \Sigma$$

A set of strings is called a *language*.

# 3 Deterministic Finite Automaton

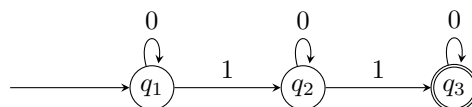
A deterministic finite automaton (DFA) is a state-machine which processes a string symbol by symbol from left to right. The automaton is in one of his *states* after processing a symbol. The machine might terminate in an *accept state* or not.

A DFA  $M = (Q, \Sigma, \delta, q, F)$

- $Q$  is a finite set of *states*
- $\Sigma$  is an alphabet
- $\delta : Q \times \Sigma \rightarrow Q$  is the *transition function*
- $q$  is an element of  $Q$  called the *start state*
- $F$  is a subset of  $Q$  which contains the *accept states*

The transition function is the logical components, it determines in which state the machine will be after processing a symbol at any state.

The following automaton processes a binary string. The start state is  $q_1$  and the only accept state is  $q_3$ . The program moves to the next state only if the symbol is 1, so it will reach  $q_3$  only if the input string contains at least two 1s.



If a DFA is in a state  $r$  and it reads the symbol  $a$ , then it will uniquely switch to the state  $\delta(r, a)$

The language of  $M$ , denoted  $L(M)$  is the set of all accepted strings by  $M$ .

$$L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$$

## 4 Regular Operations

### 4.1 Union

If  $A$  and  $B$  are two languages over the same alphabet, the union of  $A$  and  $B$  is defined as

$$A \cup B = \{w \mid w \in A \vee w \in B\}$$

### 4.2 Concatenation

If  $A$  and  $B$  are two languages over the same alphabet, the concatenation of  $A$  and  $B$  is defined as

$$AB = \{ab \mid a \in A \wedge b \in B\}$$

### 4.3 Kleene star operator

The kleene star operator can be applied to alphabets or languages. It represent the union of all  $n$ -permutations of the set.

The set  $\{0, 1\}^*$  is the set of all binary strings. If  $\Sigma$  is an alphabet,  $\Sigma^*$  is the set of all strings over  $\Sigma$

$$\Sigma^* = \lambda \cup \bigcup_{n \in \mathbb{N}} \Sigma^n$$

### 4.4 Regular language

A language is regular if it can be expressed as a regular expression, or if an automaton that accepts said language exists.

#### 4.4.1 Closure under union

*If  $A$  and  $B$  are two regular languages over the same alphabet  $\Sigma$ , there  $A \cup B$  is also regular.*

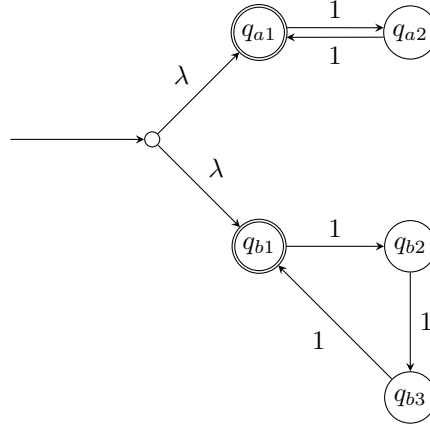
We can prove this by making an automaton that accepts both languages. Let's say that  $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$  accepts  $A$  and  $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$  accepts  $B$ . The automaton  $M = (Q, \Sigma, \delta, q, F)$  must run  $M_1$  and  $M_2$  *simultaneously*, so any state must represent the current states of  $M_1$  and  $M_2$ . This means that the states of  $M$  must represent any combination of state between  $M_1$  and  $M_2$ , meaning  $Q = Q_1 \times Q_2$ . The transition function is now in the form  $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$  where  $a \in \Sigma$ . The initial state is the state in  $Q$  which contains the initial state of  $M_1$  and  $M_2$ , namely  $(q_1, q_2)$ . Finally, the set of accept states is every tuple in  $Q_1$  containing a state in  $F_2$  or in  $Q_2$  containing a state in  $F_1$ , namely  $Q_1 \times F_2 \cup Q_2 \times F_1$ .

We can conclude that  $M = (Q_1 \times Q_2, \Sigma, \delta((r_1, r_2), a), (q_1, q_2), Q_1 \times F_2 \cup Q_2 \times F_1)$  accepts  $A \cup B$  so  $A \cup B$  is regular.

## 5 Nondeterministic Finite Automaton

Nondeterministic finite automata (NFA) are state-machines like DFAs but can change multiple states at a time by processing empty strings  $\lambda$  and when processing a symbol  $a$  may have multiple possible states to switch to. The NFA will choose the "correct" switch in order to end in an accept state, if possible.

The following automaton where  $\Sigma = \{1\}$  will end in an accept state if the input has length which is a multiple of 2 or 3.



The first switch is done by processing an empty string and the direction is chosen magically in order to end in an accept state.

A NFA is defined as  $M = (Q, \Sigma, \delta, q, F)$  where

- $Q$  is a finite set of *states*
- $\Sigma$  is an alphabet
- $\delta : Q \times \Sigma_{\lambda} \rightarrow \mathcal{P}(Q)$  is the *transition function*
- $q$  is an element of  $Q$  called the *start state*
- $F$  is a subset of  $Q$  which contains the *accept states*

## 6 Equivalence of DFAs and NFAs

Anything that can be computed by a NFA can also be computed by a DFA and vice versa.

### 6.1 DFA to NFA conversion

Let  $M = (Q, \Sigma, \delta, q, F)$  be a DFA.  $\delta$  is not a transition function of a NFA, so we need to redefine it as  $\delta'$ . Since  $\delta$  cannot process  $\lambda$ ,  $\delta'$  it is defined as

$$\delta'(r, a) = \begin{cases} \delta(r, a) & x \neq \lambda \\ \emptyset & x = \lambda \end{cases}$$

where  $r$  is a state in  $Q$  and  $a$  is a symbol in  $\Sigma_{\lambda}$ .

We can conclude that  $N = (Q, \Sigma, \delta', q, F)$ .