Automata and Formal Languages

Matteo Alberici

May 2020

Contents

1 Introduction

The theory of computation can be divided into three areas:

- **Complexity Theory**

  What makes problems computationally hard or easy?

- **Computability Theory**

  What problems can(not) computers solve?

- **Automata Theory**

  Deals with defining mathematical models of computation

2 Mathematical Notions and Terminology

2.1 Sets

A **set** is a group of unordered unique **elements** seen as a unit:

$$S = \{1,\ ciao,\ 3\}$$

The **cardinality** of a set is the number of its elements:

$$|S| = 3$$

The **empty set** $\emptyset$ is the set with no members.
A set A is a **subset** of set B if every element of A is member of B:

$$A \subseteq B$$

Moreover, A is a **proper subset** of B if A is not equal to B:

$$A \subset B$$

The **union** of two sets A and B is the set containing all the elements in A and all those in B:

$$A \cup B$$

The **intersection** of two sets A and B is the set containing the elements that are in both A and B:

$$A \cap B$$

The **complement** of set A is the set of all elements not in A:

$$\bar{A}$$

The **power set** of set A is the set of all the subsets of A:

$$2^A$$

The **Cartesian product** of sets A and B is the set of all ordered pairs wherein the first element is a member of A and the second element is a member of B:

$$A \times B$$

2.2 Sequences and Tuples

A **sequence** is an ordered list of objects:

$$(1, 2, 3)$$

A finite sequence is a **tuple**: a **k-tuple** contains k elements.
A 2-tuple is called an **ordered pair**.

2.3 Functions and Relations

A **function** is an object that sets up an input–output relationship. A function f whose input and output values are a and b is represented as follows:

$$f(a) = b$$

A function is also called a **mapping**, thus: "f maps a to b".
Given a function f, the set of its possible inputs is called the **domain**, while the set of its possible outputs is called the **range**:

$$f \ : \ D \ \rightarrow \ R$$

A function that uses all the elements of the range is **onto** the range.
If the domain of a function f is $A_1 \times ... \times A_k$, then the input is a k-tuple $(a_1, ..., a_k)$ where a_i is an **argument** to f. A function f with k arguments is a **k-ary function**, where k is the **arity** of f. An **unary function** has one argument, while a **binary function** has two arguments.
A **predicate** is a function whose range is {true, false}.
A predicate whose domain is a set of k-tuples $A \times ... \times A$ is a **k-ary relation** on A. A $2-$ary relation is called a **binary relation**.
An **equivalence relation** R captures the notion of two objects being equal in some feature and must satisfy the following conditions:

1. R is **reflexive**: xRx for every x
2. R is **symmetric**: xRy implies yRx for every x, y
3. R is **transitive**: xRy and yRz imply xRz, for every x, y, z.

2.4 Graphs

A **graph** is a set of **nodes** connected with **edges**. We can label nodes and edges obtaining a **labeled graph**.
In a **directed graph**, edges have directions.
A graph G is a **subgraph** of a graph H if the nodes and the edges of G are subsets of the ones of H. A **simple cycle** contains at least three nodes and repeats the first and last ones.
A graph is **connected** if every node has a path to the others.
A graph is a **tree** if it is connected and has no simple cycles. Trees have a node designated as **root**, while the other nodes of degree 1 are called the **leaves**.

A **path** is a sequence of nodes connected by edges. A **simple path** is a path with no **cycles**, meaning paths that start and end in the same node. A path wherein all the arrows point in the same direction as its steps is a **directed path**: a directed graph is **strongly connected** if a directed path connects every two nodes.

2.5 Strings and Languages

An **alphabet** Σ is a nonempty finite set of **symbols**:

$$\Sigma = \{a, b, c, ..., z\}$$

A **string over an alphabet** is a sequence of symbols from Σ:

$$abracadabra \text{ is a string over } \Sigma$$

Given a string w, then the **length** $|w|$ is the number of its symbols. The **empty string** ϵ is the string with length 0.
Given two strings x and y, then the **concatenation** xy is obtained by appending y to the end of x:

$$aa \circ bb = aabb,$$

where aa is the **prefix** and bb is the suffix.
A **language** L is a set of strings.

2.6 Definitions, Theorems, and Proofs

Definitions describe objects and notions we make **mathematical statements** about, expressing some property.
A **proof** is a convincing logical argument that a statement is true, while a **theorem** is a mathematical statement proved true. Statements useful only because they assist in the proof of another are called **lemmas**. Theorems determining that related statements are true are called **corollaries**.

2.6.1 Finding Proofs

Truth or falsity of statements is determined trough a mathematical proof. Given the multipart statement *"P iff Q"*, we have the following definitions:

- **Forward direction**: if P is true, then Q is true - $P \Rightarrow Q$
- **Reverse direction**: if Q is true, then P is true - $Q \Leftarrow P$
- Finally, $P \iff Q$

If a statement states that all objects of a certain type have a particular property, we must find a **counterexample** in order to prove it wrong.

2.6.2 Types of Proofs

Proof by Construction

A **proof by construction** shows that a particular type of object exists. For example, we build a **k-regular graph** if every node in the graph has degree k.

Proof by Contradiction

In a **proof by contradiction**, we assume that a theorem is false and then show that it leads to a false consequence. For example, proving that the square root of 2 is an irrational number.

Proof by Induction

A **proof by induction** shows that all elements of an infinite set have a specified property. For example, we show that an arithmetic expression computes a desired quantity for every assignment to its variables.
Given a property P, then the **basis** proves that $P(1)$ is true, while the **induction step** proves that if $P(1)$ is true, then so is $P(i+1)$. The assumption that $P(i)$ is true is the **induction hypothesis**.

3 Boolean Logic

Boolean logic is a mathematical system built around the **Boolean values** true and false, manipulated through **Boolean operations**:

- **Negation**: returns the opposite value (NOT - $\neg$)
- **Conjunction**: returns 1 if both values are 1 (AND - $\wedge$)
- **Disjunction**: returns 1 if either of the values is 1 (OR - $\vee$)
- **Exclusive or**: returns 1 if only one value is 1 (XOR - $\oplus$)
- **Equality**: returns 1 if both values have the same value ($\iff$)
- **Implication**: returns 0 iff the first operand is 1 and the second one is 0 ($\implies$)

$$
\begin{array}{ccc}
0 \wedge 0 = 0 & 0 \vee 0 = 0 & \neg 0 = 1 \\
0 \wedge 1 = 0 & 0 \vee 1 = 1 & \neg 1 = 0 \\
1 \wedge 0 = 0 & 1 \vee 0 = 1 & \\
1 \wedge 1 = 1 & 1 \vee 1 = 1 & \\
0 \oplus 0 = 0 & 0 \leftrightarrow 0 = 1 & 0 \rightarrow 0 = 1 \\
0 \oplus 1 = 1 & 0 \leftrightarrow 1 = 0 & 0 \rightarrow 1 = 1 \\
1 \oplus 0 = 1 & 1 \leftrightarrow 0 = 0 & 1 \rightarrow 0 = 0 \\
1 \oplus 1 = 0 & 1 \leftrightarrow 1 = 1 & 1 \rightarrow 1 = 1
\end{array}
$$

Figure 1: Boolean operations results

The **distributive law** for AND and OR states that:

- $P \wedge (Q \vee R) \iff (P \wedge Q) \vee (P \wedge R)$
- $P \vee (Q \wedge R) \iff (P \vee Q) \wedge (P \vee R)$

4 Regular Languages

Theory of computation uses an idealized computer called a **computational model**.

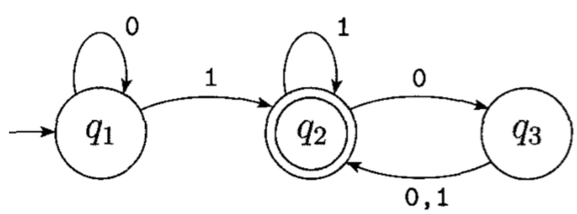
4.1 Finite Automata

The simplest computational model is the **finite automaton** (**FA**). It receives a string in input and, by processing it, either **accepts** or **rejects** it. An FA is defined by the following **5-tuple**:

$$A = (Q, \Sigma, \delta, q_0, F), \text{ where}$$

- Q is the finite set of states
- Σ is the finite input alphabet
- $\delta : Q \times \Sigma \to Q$ is the transition function
- $q_0 \in Q$ is the initial state
- $F \subseteq Q$ is the accepting state

Let's analyze the following example reading string 1101 in the machine M:



1. Start in the initial state q_1
2. Read 1, follow transition from q_1 to q_2
3. Read 1, follow transition from q_2 to q_2
4. Read 0, follow transition from q_2 to q_3
5. Read 1, follow transition from q_3 to q_2
6. Accept because M is in the accepting state q_2

A language is **regular** if an FA recognises it. Given the set A of all strings accepted by a machine M, then A is the **language of machine** M:

$$L(M) = A$$

There are three **regular operations** used to study properties of regular languages:

- **Union**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
- **Concatenation**: $AB = \{xy \mid x \in A \text{ and } y \in B\}$
- **Star**: $A* = \{x_1, x_2, ..., x_k \mid k \geq 0 \text{ and } x_i \in A\}$

4.2 Nondeterminism

In a **nondeterministic finite automaton** (**NFA**), several choices may exist for the next state:

- Multiple ways to proceed, thus the machine **splits** in copies
- No ways to proceed, thus the copy **dies**
- A copy is in an accepting state, thus the string is accepted
- An ε transition exists, thus the machine **splits immediately**

An NFA is defined by the following **5-tuple**:

$$FA = (Q, \Sigma, \delta, q_0, F), \text{ where}$$

- Q is the set of states
- Σ is the input alphabet
- $\delta : Q \times \Sigma_\varepsilon \to P(Q)$ is the **transition function**
- $q_0 \in Q$ is the **initial state**
- $F \subset Q$ is the **accepting state**

4.2.1 From NFA to FA

There exist an NFA for each FA:

1. Draw q_0 and the states reached from it along an ε transition
2. Draw at least one accepting state
3. Draw all transitions
   - No label: the transition leads to the **dead state** $\emptyset$
   - ε transition: the transition leads to the subset of states reached along it

4.3 Regular Expressions

Regular expressions describe languages and are built up using regular operations:

$$(0 \cup 1) \rightarrow (\{0\} \cup \{1\}) \rightarrow \{0,1\} = \Sigma$$
$$\Sigma 0 \rightarrow \{00, 10\}$$
$$0* \rightarrow \{0*\} \rightarrow \{\varepsilon, 0, 00, ...\}$$

R is a regular expression over an alphabet Σ if it is at least one of:

1. a, for some a in Σ
2. ε
3. $\emptyset$
4. $R_1 \cup R_2$, where R_1 and R_2 are regular expressions
5. $R_1 R_2$, where R_1 and R_2 are regular expressions
6. R_1*, where R_1 is a regular expression

5 Nonregular Languages

To prove if a language is regular or not, we use the following procedure. **pumping lemma**, which states that if L is a regular language, then there is a **pumping length** p such that, if w is a string in L of at least length p, then w may be divided into three pieces, x, y, and z, satisfying the following conditions:

1. $xy^iz \in L$ for each $i \geq 0$
2. $|y| > 0$
3. $|xy| \leq p$

We must choose a string $s \in L$ of at least length p, then show that no possible division satisfies all conditions.
Let's analyze an example with the following language:

$$L = \{w | w = 0^n 1^n \text{ for } n \geq 0\}$$

1. Let's assume L is regular
2. A pumping length p must exist such that all strings in L of at least length p can be pumped
3. Choose a string w that cannot be pumped:

$$w = 0^p 1^p \rightarrow 00...011...1$$

4. Show that no division into xyz that satisfies all conditions exists:
   (a) 3^{rd} condition: y must contain only 0s
   (b) 2^{nd} condition: y must contain at least one 0
   (c) 1^{st} condition: repeating 0s results in a string $\notin L$

6 Context-Free Languages

A **context-free grammar** (**CFG**) is a collection of **substitution rules** which are composed of a **variable**, a symbol, and a string, which again is composed of variables or **terminals**, built as follows:

1. Write the **start variable**:
2. Find a rule starting with a known variable
3. Replace the variable with the correspondent string

$$
\begin{aligned}
&A \to 0A1 \qquad & A \to 0A1 \to 00A11 \to 00B11 \to 00Z11 \\
&A \to B & A \to B \to Z \\
&B \to Z & L = \{0^n Z 1^n \mid n \geq 0\}
\end{aligned}
$$

A language generated by a CFG is a **context-free language** and it defined by the following **4-tuple**:

$$L = (V, \Sigma, R, S), \text{ where}$$

- V is the set of variables
- Σ is the set of terminals
- R is the set of rules
- $S \in V$ is the start variable

The following are some properties of CGFs:

1. Break a language into simpler languages:
    - Union:

$$
\begin{aligned}
&S_1 \to 0S_11 \mid \varepsilon \\
&S_2 \to 0S_21 \mid 01 \\
&S \to S_1 \mid S_2
\end{aligned}
$$

    - Concatenation:

$$
\begin{aligned}
&S_1 \to 0S_11 \mid \varepsilon \\
&S_2 \to 0S_21 \mid 01 \\
&S \to S_1S_2
\end{aligned}
$$

2. Draw an FA and convert it to its CFG:
   (a) Design a variable R_i for every state Q_i
   (b) Add a rule $R_i \rightarrow aR_j$ for the $a-$transition from Q_i to Q_j
   (c) Add a rule $R_i \rightarrow \varepsilon$ if Q_i is a final state
   (d) Set the variable R_0 for the initial state Q_0

A sequence of substitutions is called a **derivation**. A string is **derived ambiguously** if it has many derivations, thus a grammar is **ambiguous** if it generates at least one ambiguous string.

7 Pushdown Automata

A **pushdown automaton** (**PDA**) is an NFA equipped with a **stack** giving it an unbounded memory needed for some non-regular languages. A PDA is defined by the following **6-tuple**:

$$P = (Q, \Sigma, \Gamma, \delta, q_0, F), \text{ where}$$

- Q is the set of states
- Σ is the alphabet
- Γ is the **stack alphabet**
- $\delta : Q_i \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to P(Q_j \times \Gamma_\varepsilon)$ is the transition function, where:
  - Q_i is a state
  - Σ_ε is an input symbol
  - Γ is a stack symbol read
  - Q_j the state reached
  - Γ_ε is the new symbol on the stack
- q_0 is the initial state
- $F \subseteq Q$ is the set of accepting states

The **transition labels** are written as follows:

$$A, B \to C,$$

meaning that if A is read from the input string and B is on top of the stack, then pop B and write C.