0.1 Computational Problems

A computational problem can be modeled so that it can be solved using a computer:

- e.g. arithmetic, lexicographic ordering, salary accounting, course maintenance, ...

A representation of the problem that is more general than the solving program is easier to understand and makes it possible to analyze the problem.

A problem has *instances* (= inputs), its solution is an *algorithm*, which connects an *answer* (= output) to any instance.

An instance and its answer must be *finitely represented* (e.g. as bit strings). The number of instances, though, can be infinite.

A computational problem is a *mapping from the set of finitely represented instances to the set of finitely represented answers*.

---

Strings and Languages

Finite representation = a string over an alphabet

- An *alphabet* is a nonempty finite set of *symbols*
  - E.g., *binary alphabet* \{0, 1\} and the Latin alphabet \{a, b, ..., z\}

A *string* over an alphabet is a finite sequence of symbols from the alphabet

- E.g., 01001 and *abracadabra*

The *length* of a string \(w\), written |\(w\)|, is the number of symbols that it contains.

- E.g. |01001| = 5 \(\epsilon\) |abracadabra| = 11

The *empty string* \(\epsilon\) has length |\(\epsilon\)| = 0
• Appending strings, the **concatenation**, is their basic operation.
  • \textit{abracadabra} = \textit{abracadabra}
  • \( x = 01, y = 10 \rightarrow xx = 0101, xy = 0110, yy = 1010 \) \( ja \ yx = 1001 \)
  • For all \( w \) it holds that \( \varepsilon w = \varepsilon w = w \)
  • For all \( x, y \) it holds that \( |xy| = |x| + |y| \)
  • If \( w = xy \), then \( x \) is a **prefix** of \( w \) and \( y \) is its **suffix**
  • All strings of the alphabet \( \Sigma \) are denoted by \( \Sigma^* \)
    • E.g. \( \Sigma = \{ 0, 1 \} \rightarrow \Sigma^* = \{ \varepsilon, 0, 1, 00, 01, 10, 11, \ldots \} \)
  • Other notation: \( \Sigma^k \) and \( \Sigma^* = \bigcup_{k \in \mathbb{N}} \Sigma^k \)

**Decision Problems**

• A computational problem \( \pi \) is hence a mapping
  \[
  \pi : \Sigma^* \rightarrow \Gamma^*,
  \]
  where \( \Sigma \) and \( \Gamma \) are alphabets

• **Decision problems** are an important subclass; in them the answer to an instance of the problem is simply **Yes** or **No**
  • I.e., they have the form \( \pi : \Sigma^* \rightarrow \{ 0, 1 \} \)

  • For every decision problem \( \pi \) there is the set of those instances for which the answer is **Yes**:
    \[
    A_\pi = \{ x \in \Sigma^* \mid \pi(x) = 1 \}
    \]
• The other way around: For every set of strings $A$ there exists a decision problem $\pi_A: \Sigma^* \rightarrow \{0, 1\}$,

$$\pi_A(x) = \begin{cases} 1, & \text{if } x \in A \\ 0, & \text{if } x \notin A \end{cases}$$

• The set of strings $A \subseteq \Sigma^*$ is called a (formal) language of the alphabet $\Sigma$.
• The respective decision problem $\pi_A$ is known as the recognition problem of the language $A$.
• We can treat formal languages and decision problems as equals.

---

**Definition 4.12**

• Let $A$ and $B$ be two sets and $f$ a function from $A$ to $B$.
• We say that $f$ is one-to-one (injection) if it never maps two different elements to the same place — that is, $f(a) \neq f(b)$ whenever $a \neq b$.
• We say that $f$ is onto (surjection) if it hits every element of $B$ — that is, if for every $b \in B$ there is an $a \in A$ such that $f(a) = b$.
• We say that $A$ and $B$ are the same size if there is a one-to-one, onto function $f: A \rightarrow B$.
• A function that is both one-to-one and onto is called a correspondence (bijection).
  • In a correspondence every element of $A$ maps to a unique element of $B$ and each element of $B$ has a unique element of $A$ mapping to it.
  • A correspondence is simply a way of pairing the elements of $A$ with the elements of $B$. 
Solvability of Computational Problems

- We say that program \( P(x) \) solves the computational problem \( \pi \), if it outputs for each input \( x \) the value \( \pi(x) \).
- Can all possible computational problems be solved by programs (computers)?

**Definition 4.14:** A set \( A \) is **countable** if either it is finite or it has the same size as \( \mathbb{N} \).

- An infinite set that is not countable is **uncountable**

---

**Theorem 0.1** Let \( \Sigma \) be an arbitrary alphabet. The set of strings over \( \Sigma, \Sigma^* \), is countable.

**Proof.** Let \( \Sigma = \{ a_1, a_2, ..., a_n \} \). Let us fix an "alphabetical ordering" for the symbols; let it be \( a_1 < a_2 < ... < a_n \).

The strings belonging to \( \Sigma^* \) can now be output in **lexicographic (or canonical) order**:

1. First output strings of length 0, then those of length 1, after which those of length 2, and so forth.
2. Within each length group the strings are given in the dictionary order as determined by the chosen alphabetical order.
Then the correspondence $f: \mathbb{N} \to \Sigma^*$ is:

- $0 \mapsto \varepsilon$
- $1 \mapsto a_1$
- $2 \mapsto a_2$
- $\vdots$
- $n \mapsto a_n$
- $n+1 \mapsto a_1a_1$
- $n+2 \mapsto a_1a_2$
- $\vdots$

- $2n \mapsto a_1a_n$
- $2n+1 \mapsto a_2a_1$
- $\vdots$
- $3n \mapsto a_2a_n$
- $\vdots$
- $n^2+n \mapsto a_n a_n$
- $n^2+n+1 \mapsto a_1a_1a_1$
- $n^2+n+2 \mapsto a_1a_1a_2$
- $\vdots$

\[\square\]

**Theorem 0.2** The set of decision problems over any alphabet $\Sigma$ is uncountable.

**Proof.** Let $\Pi$ denote the collection of all decision problems over $\Sigma$: $\Pi = \{\pi \mid \pi$ is a mapping $\Sigma^* \to \{0, 1\}\}.$

Let us assume that $\Pi$ is countable; i.e., there exists an enumeration that covers all elements of $\Pi$:

$\Pi = \{\pi_0, \pi_1, \pi_2, \ldots\}.$

Let the strings belonging to $\Sigma^*$, given in the lexicographic ordering of Theorem 1.1, be $x_0, x_1, x_2, \ldots$
Let us compose a new decision problem $\xi: \Sigma^* \rightarrow \{0, 1\}$:

$$\xi(x_i) = \begin{cases} 1, & \text{if } \pi_i(x_i) = 0 \\ 0, & \text{if } \pi_i(x_i) = 1 \end{cases}$$

Because $\Pi$ covers all decision problems over $\Sigma$, it must be that $\xi \in \Pi$. Hence, $\xi = \pi_k$ for some $k \in \mathbb{N}$. Then

$$\xi(x_k) = \begin{cases} 1, & \text{if } \pi_k(x_k) = \xi(x_k) = 0 \\ 0, & \text{if } \pi_k(x_k) = \xi(x_k) = 1 \end{cases}$$

This is a contradiction. Thus, our assumption ($\Pi$ is countable) cannot hold. Hence, $\Pi$ must be uncountable.

---

- This type of proof is known as Cantor's diagonalization method.
- In the end, e.g., Java programs are just strings over the alphabet ASCII. By Theorem 0.1 there exist only a countable set of them.
- However, by Theorem 0.2 the set of computational problems is uncountable.
- Therefore, out of all computational problems only a miniscule part can be solved using Java programs.
- The problem is the same for all programming languages.
- Unsolvable problems include also interesting and practical problems.
Georg Cantor (1845–1918)

- Born in St. Petersburg
- To Frankfurt 1856
- Ph.D. (Berlin) 1867
- Halle 1869-
  - Privatdozent
  - Prof. 1872
- Countability of rational numbers 1873
- Set theory 1874
- Continuum hypothesis 1878

AUTOMATA AND LANGUAGES

1.1 Finite Automata

- A system of computation that only has a finite number of possible states can be modeled using a finite automaton
- A finite automaton is often illustrated as a state diagram
Definition 1.5: Finite Automaton

- A finite automaton is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$, where
  - $Q$ is a finite set called the states,
  - $\Sigma$ is a finite set called the alphabet,
  - $\delta: Q \times \Sigma \rightarrow Q$ is the transition function,
  - $q_0 \in Q$ is the start state, and
  - $F \subseteq Q$ is the set of (accepting) final states.

- A machine $M$ accepts the string $w = w_1w_2...w_n \in \Sigma^n$ if a sequence of states $r_0, r_1, ..., r_n$ in $Q$ exists s.t.
  - $r_0 = q_0$,
  - $\delta(r_i, w_{i+1}) = r_{i+1}$, $i = 0, ..., n-1$,
  - $r_n \in F$.

- The language recognized by $M$ is
  $$L(M) = \{ w \in \Sigma^* | M \text{ accepts } w \}$$

- A language is called a regular language, if some finite automaton recognizes it.

- Basic operations on languages $A$ and $B$ are
  - **Union** $A \cup B = \{ x | x \in A \lor x \in B \}$,
  - **Concatenation** $A \cdot B = \{ xy | x \in A \land y \in B \}$ and
  - **(Kleene) Star (closure)** $A^* = \{ x_1x_2...x_r | k \geq 0 \land x_i \in A \forall i \}$
Properties of Regular Languages

**Theorem 1.25** The class of regular languages is closed under the union operation.

In other words, if $A_1$ and $A_2$ are regular languages, so is $A_1 \cup A_2$.

**Theorem 1.26** The class of regular languages is closed under the concatenation operation.

DFA = Deterministic Finite Automaton

---

1.1.1 Minimization of DFAs

- Two automata that recognize exactly the same language are equivalent with each other.
- A finite automaton is minimal if it has the smallest number of states among equivalent automata.
- An automaton that has more states than in an equivalent minimal automaton is called redundant.
- Algorithms producing automata do not always generate a minimal automaton.
- Handling a minimal automaton is more efficient than that of a redundant automaton.
**Algorithm MINIMIZE**

**Input:** DFA $M = (Q, \Sigma, \delta, q_0, F)$.
1. Remove all states of $M$ that are unreachable from the start state.
2. Construct the following undirected graph $G$ whose nodes are the states of $M$.
3. Place an edge in $G$ connecting every accept state with every nonaccept state. Add additional edges as follows.
4. Repeat until no new edges are added to $G$:
   1. For every pair $q, r \in Q$, $q \neq r$, and every $a \in \Sigma$: add the edge $(q, r)$ to $G$ if $(\delta(q, a), \delta(r, a))$ is an edge of $G$.
   2. For each state $q \in Q$ let $[q]$ be the collection of states
      
      $[q] = \{ q \} \cup \{ r \in Q | \text{no edge joins } q \text{ and } r \text{ in } G \}.$

5. Form a new DFA $M' = (Q', \Sigma, \delta', q_0', F')$, where
   - $Q' = \{ [q] | q \in Q \}$ (removing doubles)
   - $\delta'([q], a) = \delta(q, a)$, for every $q \in Q$ and $a \in \Sigma,$
   - $q_0' = [q_0]$ and
   - $F' = \{ [q] | q \in F \}$.
6. Output $M'$. 
The End Result

- An automaton $M'$ that is equivalent with the input automaton $M^*$, such that it has the minimum number of states.
- Automaton $M'$ is unique (up to the naming of the states).
1.2 Nondeterministic Finite Automata (NFAs)

- In an NFA a state can have many possible alternative transitions with the same symbol of the alphabet.
- Also $\epsilon$-transitions are allowed.
- Implementing nondeterministic behavior is not straightforward (though possible), but as a modeling tool it is quite useful.
- Via NFAs we can connect DFAs and regular expressions.