
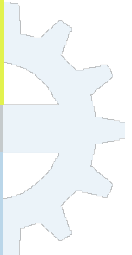


OHJ-2306 **Fall 2009**
Introduction to Theoretical Computer Science

 TAMPERE UNIVERSITY OF TECHNOLOGY
Department of Software Systems

OHJ-2306 Introduction to Theoretical Computer Science, Fall 2009 10.9.2009

2



Organization & timetable


Lectures: prof. Tapio Elomaa

- P1: Tue 14-16 TC 103 and Thu 14-16 TC 133
- P2: Tue 14-16 TB 219 and Thu 12-14 TB 224
- Sept. 8 – Oct. 15, Oct. 27 – Dec. 3
 - Ph.D. Jussi Kujala gives the lectures in the second week (Tapio Elomaa in Prague)

Weekly exercises 24.9. →

- Ph.D. Jussi Kujala, Thu 10-12 TC 128

Exam: Tue Dec. 15, 2009, 9-12 AM
Assessment: 30+8=38
{elomaa, kujala2}@cs.tut.fi

 TAMPERE UNIVERSITY OF TECHNOLOGY
Department of Software Systems

OHJ-2306 Introduction to Theoretical Computer Science, Fall 2009 10.9.2009

Weekly exercises

- It is **most advisable** to take part in the weekly exercises
- Being ready to present your solution to problem yields one mark
- Each weekly exercise session has approx. 6 problems → altogether you can collect approx. $6 \times 10 = 60$ marks

Marks	Extra points
20% (c. 12)	1
30% (c. 18)	2
40% (c. 24)	3
...	...
80% (c. 48)	7
90% (c. 54)	8

Grading

- You can pass the course with grade 5/5 by taking the exam alone, but
 - The exam (max 30 p.) is not the easiest of them all
 - Therefore it is advisable to participate in the weekly exercises from which you can earn extra points (max 8 p.)
 - By independently solving the problems you learn the seemingly difficult material of the course
- The course grade will most probably be determined according to the following table:

points	15	18	21	24	27
grade	1	2	3	4	5

Course Material

- The course text book in Fall 2009 is

Michael Sipser: *Introduction to the Theory of Computation*,
Second Ed. (International Ed.), Thomson, 2006

- Almost any text book on this topic covers the material that we will go through
- The slides will appear into the Web weekly
 - <http://www.cs.tut.fi/kurssit/OHJ-2300/2306.html>
 - <http://www.cs.tut.fi/~elomaa/teach/2306.html/>
- The exam is based on the lectures



Timetable for Lectures

- 1. Introduction (1)**
- 2. Recap: automata, grammars and languages (2-3)**
 - Regular languages
 - Context-free grammars
- 3. Computability theory (4-7)**
 - Universal models of computation
 - Solvability / decidability
 - Reducibility
 - Advanced topics
- 4. Complexity theory (8-12)**
 - Time complexity
 - Space complexity
 - Practical solvability
 - Advanced topics





0. Introduction

- This course offers an introduction to the mathematical and theoretical background of computer science
- Such information is an integral part of the general knowledge of a computer scientist
- The aim is to gain a basic understanding of what kinds of problems can in principle be solved using a computer
- Even more important is to observe which of the decidable problems can be solved *efficiently* by a computer program
- The results that we cover in this course are fundamental in the sense that the increasing efficiency of computers in the years to come will not deem these results insignificant



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 $\{a, b, \dots, z\}$
- 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. $|01001| = 5$ ja $|abracadabra| = 11$
- The *empty string* ε has length $|\varepsilon| = 0$



- Appending strings, the *concatenation*, is their basic operation.
 - **abra°cadabra = abracadabra**
 - $x = 01, y = 10 \rightarrow xx = 0101, xy = 0110, yy = 1010$ ja $yx = 1001$
 - For all w it holds that $w\varepsilon = \varepsilon w = w$
 - 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 Σ are denoted by Σ^*
 - E.g. $\Sigma = \{0, 1\} \rightarrow \Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, \dots\}$
- Other notation: Σ^k and $\Sigma^* = \bigcup_{k \in \mathbf{N}} \Sigma^k$



Decision Problems

- A computational problem π is hence a mapping

$$\pi: \Sigma^* \rightarrow \Gamma^*,$$

where Σ and Γ 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 π 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 Σ
- The respective decision problem π_A is known as the recognition problem of the language A
- We can treat formal languages and decision problems as equals



Solvability of Computational Problems

- We say that program $P(x)$ solves the computational problem π , if it outputs for each input x the value $\pi(x)$
- Can all possible computational problems be solved by programs (computers)?
- An infinite set X is *countable* if there exists a bijective function $f: \mathbb{N} \rightarrow X$.
- Also finite sets are countable
 - Bijection = injection (one-to-one) + surjection (onto)
 - Injection: $f(x_1) = f(x_2) \Rightarrow x_1 = x_2$
 - Surjection: $f(A) = B$
- An infinite set that is not countable is *uncountable*



Theorem 0.1 Let Σ be an arbitrary alphabet. The set of strings over Σ , Σ^* , is countable.

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

The strings belonging to Σ^* 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 bijection $f: \mathbb{N} \rightarrow \Sigma^*$ is

$$\begin{aligned} 0 &\mapsto \varepsilon \\ 1 &\mapsto a_1 \\ 2 &\mapsto a_2 \\ &\vdots \\ n &\mapsto a_n \\ n+1 &\mapsto a_1a_1 \\ n+2 &\mapsto a_1a_2 \\ &\vdots \end{aligned}$$

$$\begin{aligned} 2n &\mapsto a_1a_n \\ 2n+1 &\mapsto a_2a_1 \\ &\vdots \\ 3n &\mapsto a_2a_n \\ &\vdots \\ n^2+n &\mapsto a_na_n \\ n^2+n+1 &\mapsto a_1a_1a_1 \\ n^2+n+2 &\mapsto a_1a_1a_2 \\ &\vdots \end{aligned}$$

□



Theorem 0.2 *The set of decision problems over any alphabet Σ is uncountable.*

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

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

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

Let the strings belonging to Σ^* , given in the lexicographic ordering of Theorem 1.1, be x_0, x_1, x_2, \dots



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 Π covers all decision problems over Σ , 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 (Π is countable) cannot hold. Hence, Π must be uncountable. \square



- 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)

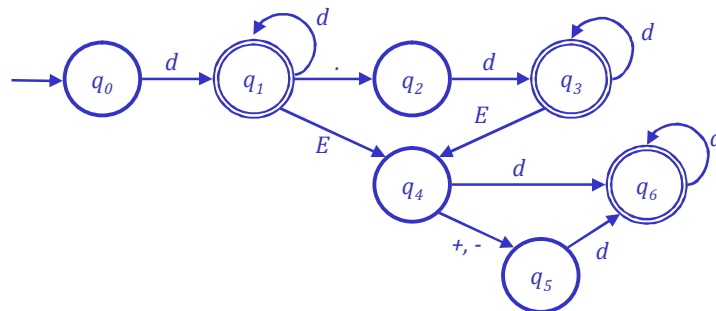
- Born in St. Petersburg
- To Frankfurt 1856
- Ph.D. (Berlin) 1867
- Halle 1869-
 - Privatdozent
 - Prof. 1872
- Countability of rational numbers 1873
- Set theory 1874
- Continuum hypothesis 1878



1. RECAP

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*



Definitions 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**,
 - Σ 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 $w = w_1w_2\dots w_n \in \Sigma^n$ if a sequence of states r_0, r_1, \dots, r_n in Q exists s.t.
 - $r_0 = q_0$
 - $\delta(r_i, w_{i+1}) = r_{i+1}, i = 0, \dots, n-1,$
 - $r_n \in F.$



- The **language recognized** by M is

$$L(M) = \{ w \in \Sigma^* \mid M \text{ accepts } w \}$$
- A language is called a **regular language**, if some finite automaton recognizes
- Basic operations on languages A and B are
 - **Union** $A \cup B = \{ x \mid x \in A \vee x \in B \},$
 - **Concatenation** $A \circ B = \{ xy \mid x \in A \wedge y \in B \}$ and
 - **(Kleene) Star (closure)** $A^* = \{ x_1x_2\dots x_k \mid k \geq 0 \wedge x_i \in A \forall i \}$





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