Theory of Computer Science

C1. Turing Machines as Formal Model of Computation

Gabriele Röger

University of Basel

April 10, 2024

C1.1 Hilbert's 10th Problem

C1.2 Church-Turing Thesis

C1.3 Encoding

C1.4 Summary

Overview: Course

contents of this course:

A. background ✓
- ▷ mathematical foundations and proof techniques

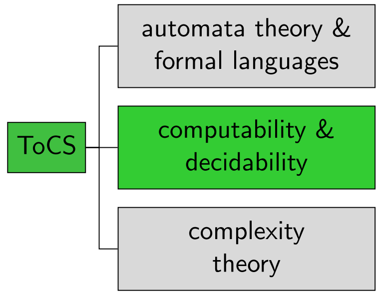
B. automata theory and formal languages ✓
- ▷ What is a computation?

C. Turing computability
- ▷ What can be computed at all?

D. complexity theory
- ▷ What can be computed efficiently?

E. more computability theory
- ▷ Other models of computability

Content of the Course



Main Question

Main question in this part of the course:

What can be computed by a computer?

C1.1 Hilbert's 10th Problem

Algorithms

- Informally, an algorithm is a collection of simple instructions for carrying out some task.
- Long history in mathematics since ancient times: descriptions of algorithms e. g. for finding prime numbers or the greatest common divisor.
- A formal notion of an algorithm itself was not defined until the 20th century.

Hilbert's 10th Problem

Around 1900 David Hilbert (German mathematician) formulated 23 mathematical problems as challenge for the 20th century.

Hilbert's 10th problem

Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients:
To devise a process according to which it can be determined in a finite number of operations whether the equation is solvable in rational integers.

What does this mean?

Diophantine Equations

- A polynomial is a sum of terms where each term is a product of a constant (the coefficient) and certain variables.
  e. g. $6x^3yz^2 + 3xy^2 - x^3 - 10$
- A polynomial equation is an equation $p = 0$, where p is a polynmial. A solutions of the equation is called a root of p.
  e. g. $6x^3yz^2 + 3xy^2 - x^3 - 10$ has a root $x = 5, y = 3, z = 0$.
- Diophantine equations are polynomial equations, where only integral roots (assigning only integer values to the variables) count as solutions.

Hilbert's 10th Problem

Hilbert's 10th problem
Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: To devise a process according to which it can be determined in a finite number of operations whether the equation is solvable in rational integers.

☞ Specify an algorithm that takes a polynomial with integer coefficients as input and outputs whether it has an integral root.

There is no such algorithm!
(implication of Matiyasevich's theorem from 1970)

C1.2 Church-Turing Thesis

Formal Notion of Algorithm?

- What is an algorithm?
  - intuitive model of algorithm (cookbook recipe)
  - vs. algorithm in modern programming language
  - vs. formal mathematical models
- Proving that no algorithm exists requires a clear notion of algorithm.

Church-Turing Thesis

Church-Turing Thesis
All functions that can be computed in the intuitive sense can be computed by a Turing machine.

- cannot be proven (why not?)
- but there is significant evidence such as equivalence of TMs and different register machines:
  - Counter machine: concept of registers
  - Random-access machine (RAM): adds indirect addressing
  - Random-access stored-program machines: related to the von Neumann architecture (very close to modern computer systems)

What about the Infinite Tape?

- Turing Machines have access to infinite storage.
- Computer systems do not.
- However: A halting (in particular: accepting) computation of a TM can only use a finite part of the tape.
- If a problem is undecidable, we cannot solve it with a computer, no matter how much memory we provide.

Turing Completeness

Church-Turing Thesis
All functions that can be computed in the intuitive sense can be computed by a Turing machine.

Vice versa:

We say that a programming language is Turing-complete to express that it can compute everything a Turing machine can.

- We can show Turing completeness by showing that with the programming language we can simulate any Turing machine.

Back to Hilbert's Problem

The corresponding formal problem (= language) is

$$D = \{p \mid p \text{ is a polynomial with an integral root}\}$$

Formal way to say that "there is no algorithm for this problem":

D is not Turing-decidable.

C1.3 Encoding

Finite Structures as Strings

- Turing machines take words (= strings) as input and can only represent strings on their tape.
- Is this a limitation?
  - Not really!
  - Computers also internally operate on binary numbers (words over $\{0, 1\}$).
  - We just need to define how a string encodes a certain structure e. g. how does a file of 0s and 1s specify an image?
  - We will have a look at two examples:
    - Example 1: Encoding of pairs of numbers
    - Example 2: Encoding of Turing machines

Encoding and Decoding: Binary Encode

Consider the function $encode : \mathbb{N}_0^2 \to \mathbb{N}_0$ with:

$$encode(x, y) := \binom{x + y + 1}{2} + x$$

- *encode* is known as the Cantor pairing function
- *encode* is computable
- *encode* is bijective

	$x = 0$	$x = 1$	$x = 2$	$x = 3$	$x = 4$
$y = 0$	0	2	5	9	14
$y = 1$	1	4	8	13	19
$y = 2$	3	7	12	18	25
$y = 3$	6	11	17	24	32
$y = 4$	10	16	23	31	40

Encoding and Decoding: Binary Decode

Consider the inverse functions
$decode_1 : \mathbb{N}_0 \to \mathbb{N}_0$ and $decode_2 : \mathbb{N}_0 \to \mathbb{N}_0$ of *encode*:

$$decode_1(encode(x, y)) = x$$
$$decode_2(encode(x, y)) = y$$

- $decode_1$ and $decode_2$ are computable

Turing Machines as Inputs

- We will at some point consider problems that have Turing machines as their input.

  $\rightsquigarrow$ "programs that have programs as input": cf. compilers, interpreters, virtual machines, etc.
- We have to think about how we can encode arbitrary Turing machines as words over a fixed alphabet.
- We use the binary alphabet $\Sigma = \{0, 1\}$.
- As an intermediate step we first encode over the alphabet $\Sigma' = \{0, 1, \#\}$.

Encoding a Turing Machine as a Word (1)

Step 1: encode a Turing machine as a word over $\{0, 1, \#\}$

Reminder: Turing machine $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}} \rangle$

Idea:

- input alphabet Σ should always be $\{0, 1\}$
- enumerate states in Q and symbols in Γ and consider them as numbers $0, 1, 2, \dots$
- blank symbol always receives number 2
- start state always receives number 0, accept state number 1 and reject state number 2

  (we can special-case machines where the start state is the accept or reject state)

Then it is sufficient to only encode δ explicitly:

- Q: all states mentioned in the encoding of δ
- $\Gamma = \{0, 1, \Box, a_3, a_4, \dots, a_k\}$, where k is the largest symbol number mentioned in the δ-rules

Encoding a Turing Machine as a Word (2)

encode the rules:

- Let $\delta(q_i, a_j) = \langle q_{i'}, a_{j'}, D \rangle$ be a rule in δ, where the indices i, i', j, j' correspond to the enumeration of states/symbols and $D \in \{\mathrm{L}, \mathrm{R}\}$.
- encode this rule as

  $w_{i,j,i',j',D} = \#\#\mathit{bin}(i)\#\mathit{bin}(j)\#\mathit{bin}(i')\#\mathit{bin}(j')\#\mathit{bin}(m)$,

  where $m = \begin{cases} 0 & \text{if } D = \mathrm{L} \\ 1 & \text{if } D = \mathrm{R} \end{cases}$
- For every rule in δ, we obtain one such word.
- All of these words in sequence (in arbitrary order) encode the Turing machine.

Encoding a Turing Machine as a Word (3)

Step 2: transform into word over $\{0, 1\}$ with mapping

$0 \mapsto 00$

$1 \mapsto 01$

$\# \mapsto 11$

Turing machine can be reconstructed from its encoding.
How?

Encoding a Turing Machine as a Word (4)

Example (step 1)

$\delta(q_0, a_3) = \langle q_3, a_2, R \rangle$ becomes ##0#11#11#10#1
$\delta(q_3, a_1) = \langle q_1, a_0, L \rangle$ becomes ##11#1#1#0#0

Example (step 2)

##0#11#11#10#1##11#1#1#0#0
111100110101110101110100110111110101110111011100110 0

Exercise: Encoding of TMs (slido)

What would be the encoding of a transition
$\delta(q_0, a_0) = (q_1, a_2, L)$ as word over $\{0, 1\}$?

Turing Machine Encoded by a Word

goal: function that maps any word in $\{0, 1\}^*$ to a Turing machine
problem: not all words in $\{0, 1\}^*$ are encodings of a Turing machine

solution: Let $\widehat{M}$ be an arbitrary fixed deterministic Turing machine (for example one that always immediately stops). Then:

Definition (Turing Machine Encoded by a Word)

For all $w \in \{0, 1\}^*$:

$$M_w = \begin{cases} M' & \text{if } w \text{ is the encoding of some DTM } M' \\ \widehat{M} & \text{otherwise} \end{cases}$$

Notation for Encoding

- Most of the time, we will not consider a particular encoding of non-string objects.
- For a single object O, we will just write $\langle\langle O \rangle\rangle$ to denote some suitable encoding of O as a string.
- For several objects $O_1, \dots, O_n$, we write $\langle\langle O_1, \dots, O_n \rangle\rangle$ for their encoding into a single string.
- In the high-level description of a TM we can refer to them as the objects they are because on the lower level the TM can be programmed to handle the encoded representation accordingly.

Example

$$L = \{\langle\!\langle G \rangle\!\rangle \mid G \text{ is a connected undirected graph}\}$$

We describe a TM that recognizes L:

On input $\langle\!\langle G \rangle\!\rangle$, the encoding of a undirected graph G:

1. Select the first node of G and mark it.
2. Repeat until no more nodes are marked:
   For each node in G, mark it if it is adjacent to a node that is already marked.
3. Scan all the nodes of G to determine whether they are all marked. If yes, accept, otherwise reject.

Implicit (lower-level detail): If the input does not encode an undirected graph, directly reject.

C1.4 Summary

Summary

- main question: what can a computer compute?
- approach: investigate formal models of computation
  $\rightarrow$ deterministic Turing machines
- Based on the (existing evidence for the) Church-Turing thesis, we will describe the behaviour of Turing machines on a higher abstraction level (such as pseudo-code).
- The formal restriction of TMs to strings is not a practical limitation but can be handled with suitable encodings.