Theory of Computer Science

D1. Turing-Computability

Gabriele Röger

University of Basel

April 11, 2018

Overview: Course

contents of this course:

- logic ✓
  - ▷ How can knowledge be represented? How can reasoning be automated?
- automata theory and formal languages ✓
  - ▷ What is a computation?
- computability theory
  - ▷ What can be computed at all?
- complexity theory
  - ▷ What can be computed efficiently?

Main Question

Main question in this part of the course:

What can be computed by a computer?

Overview: Computability Theory

- Computability
  - Imperative Models of Computation
    - Turing-Computability
    - LOOP- and WHILE-Computability
    - GOTO-Computability
  - Functional Models of Computation
    - Primitive and μ-Recursion
  - Undecidable Problems
    - (Semi-)Decidability
    - Halting Problem
    - Reductions
    - Rice's Theorem
    - Other Problems

Further Reading (German)

Literature for this Chapter (German)

Theoretische Informatik – kurz gefasst
by Uwe Schöning (5th edition)

- Chapter 2.1
- Chapter 2.2

Further Reading (English)

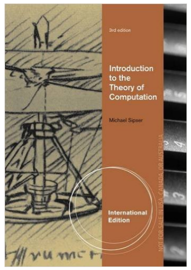
Literature for this Chapter (English)

Introduction to the Theory of Computation
by Michael Sipser (3rd edition)

- Chapter 3.1

Notes:

- Sipser does not cover all topics we do.
- His definitions differ slightly from ours.



D1.1 Computations

Computation

What is a computation?

- intuitive model of computation (pen and paper)
- vs. computation on physical computers
- vs. formal mathematical models

In the following chapters we investigate
models of computation for partial functions $f : \mathbb{N}_0^k \rightarrow_p \mathbb{N}_0$.

- no real limitation: arbitrary information
  can be encoded as numbers

German: Berechnungsmodelle

Formal Models of Computation

Formal Models of Computation

- Turing machines
- LOOP, WHILE and GOTO programs
- primitive recursive and μ-recursive functions

In the next chapters we will

- get to know these models and
- compare them according to their power.

German: Mächtigkeit

Church-Turing Thesis

Church-Turing Thesis

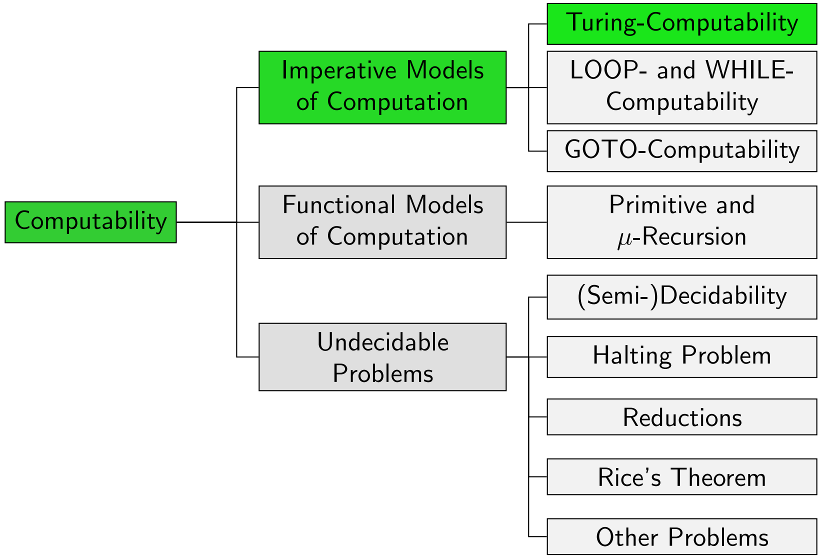
All functions that can be computed in the intuitive sense
can be computed by a Turing machine.

German: Church-Turing-These

- cannot be proven (why not?)
- but we will collect evidence for it

D1.2 Turing-Computable Functions

Overview: Computability Theory



Reminder: Deterministic Turing Machine (DTM)

Definition (Deterministic Turing Machine)

A deterministic Turing machine (DTM) is given by a 7-tuple $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \Box, E \rangle$ with:

- Q finite, non-empty set of states
- $\Sigma \neq \emptyset$ finite input alphabet
- $\Gamma \supset \Sigma$ finite tape alphabet
- $\delta : (Q \setminus E) \times \Gamma \to Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}, \mathrm{N}\}$ transition function
- $q_0 \in Q$ start state
- $\Box \in \Gamma \setminus \Sigma$ blank symbol
- $E \subseteq Q$ end states

Reminder: Configurations and Computation Steps

How do Turing Machines Work?

- configuration: $\langle \alpha, q, \beta \rangle$ with $\alpha \in \Gamma^*$, $q \in Q$, $\beta \in \Gamma^+$
- one computation step: $c \vdash c'$ if one computation step can turn configuration c into configuration c'
- multiple computation steps: $c \vdash^* c'$ if 0 or more computation steps can turn configuration c into configuration c' ($c = c_0 \vdash c_1 \vdash c_2 \vdash \cdots \vdash c_{n-1} \vdash c_n = c'$, $n \geq 0$)

(Definition of $\vdash$, i.e., how a computation step changes the configuration, is not repeated here. $\rightsquigarrow$ Chapter C8)

Computation of Functions?

How can a DTM compute a function?

- "Input" x is the initial tape content
- "Output" $f(x)$ is the tape content (ignoring blanks at the left and right) when reaching an end state
- If the TM does not stop for the given input, $f(x)$ is undefined for this input.

Which kinds of functions can be computed this way?

- directly, only functions on words: $f : \Sigma^* \to_{\mathrm{p}} \Sigma^*$
- interpretation as functions on numbers $f : \mathbb{N}_0^k \to_{\mathrm{p}} \mathbb{N}_0$: encode numbers as words

Turing Machines: Computed Function

Definition (Function Computed by a Turing Machine)

A DTM $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \square, E \rangle$ computes the (partial) function $f : \Sigma^* \to_p \Sigma^*$ for which:

for all $x, y \in \Sigma^*$: $f(x) = y$ iff $\langle \varepsilon, q_0, x \rangle \vdash^* \langle \square \dots \square, q_e, y\square \dots \square \rangle$

with $q_e \in E$. (special case: initial configuration $\langle \varepsilon, q_0, \square \rangle$ if $x = \varepsilon$)

German: DTM berechnet f

- What happens if symbols from $\Gamma \setminus \Sigma$ (e. g., $\square$) occur in y?
- What happens if the read-write head is not on the first symbol of y at the end?
- Is f uniquely defined by this definition? Why?

Turing-Computable Functions on Words

Definition (Turing-Computable, $f : \Sigma^* \to_p \Sigma^*$)

A (partial) function $f : \Sigma^* \to_p \Sigma^*$ is called Turing-computable if a DTM that computes f exists.

German: Turing-berechenbar

Encoding Numbers as Words

Definition (Encoded Function)

Let $f : \mathbb{N}_0^k \to_p \mathbb{N}_0$ be a (partial) function.
The encoded function f^{code} of f is the partial function $f^{\text{code}} : \Sigma^* \to_p \Sigma^*$ with $\Sigma = \{0, 1, \#\}$ and $f^{\text{code}}(w) = w'$ iff

- there are $n_1, \dots, n_k, n' \in \mathbb{N}_0$ such that
- $f(n_1, \dots, n_k) = n'$,
- $w = bin(n_1)\#\dots\#bin(n_k)$ and
- $w' = bin(n')$.

Here $bin : \mathbb{N}_0 \to \{0, 1\}^*$ is the binary encoding (e. g., $bin(5) = 101$).

German: kodierte Funktion
Example: $f(5, 2, 3) = 4$ corresponds to $f^{\text{code}}(101\#10\#11) = 100$.

Turing-Computable Numerical Functions

Definition (Turing-Computable, $f : \mathbb{N}_0^k \to_p \mathbb{N}_0$)

A (partial) function $f : \mathbb{N}_0^k \to_p \mathbb{N}_0$ is called Turing-computable if a DTM that computes f^{code} exists.

German: Turing-berechenbar

D1.3 Examples

Example: Turing-Computable Functions (1)

Example

Let $\Sigma = \{a, b, \#\}$.
The function $f : \Sigma^* \to_p \Sigma^*$ with $f(w) = w\#w$ for all $w \in \Sigma^*$ is Turing-computable.

$\rightsquigarrow$ blackboard

Example: Turing-Computable Functions (2)

Example

The following numerical functions are Turing-computable:

- $succ : \mathbb{N}_0 \to_p \mathbb{N}_0$ with $succ(n) := n + 1$
- $pred_1 : \mathbb{N}_0 \to_p \mathbb{N}_0$ with $pred_1(n) := \begin{cases} n-1 & \text{if } n \geq 1 \\ 0 & \text{if } n = 0 \end{cases}$
- $pred_2 : \mathbb{N}_0 \to_p \mathbb{N}_0$ with $pred_2(n) := \begin{cases} n-1 & \text{if } n \geq 1 \\ \text{undefined} & \text{if } n = 0 \end{cases}$

$\rightsquigarrow$ blackboard/exercises

Example: Turing-Computable Functions (3)

Example

The following numerical functions are Turing-computable:

- $add : \mathbb{N}_0^2 \to_p \mathbb{N}_0$ with $add(n_1, n_2) := n_1 + n_2$
- $sub : \mathbb{N}_0^2 \to_p \mathbb{N}_0$ with $sub(n_1, n_2) := \max\{n_1 - n_2, 0\}$
- $mul : \mathbb{N}_0^2 \to_p \mathbb{N}_0$ with $mul(n_1, n_2) := n_1 \cdot n_2$
- $div : \mathbb{N}_0^2 \to_p \mathbb{N}_0$ with $div(n_1, n_2) := \begin{cases} \left\lceil \frac{n_1}{n_2} \right\rceil & \text{if } n_2 \neq 0 \\ \text{undefined} & \text{if } n_2 = 0 \end{cases}$

$\rightsquigarrow$ sketch?

D1.4 Summary

Summary

- main question: what can a computer compute?
- approach: investigate formal models of computation
- first: deterministic Turing machines
- Turing-computable function $f : \Sigma^* \rightarrow_p \Sigma^*$:
  there is a DTM that transforms every input $w \in \Sigma^*$ into the output $f(w)$ (undefined if DTM does not stop or stops in invalid configuration)
- Turing-computable function $f : \mathbb{N}_0^k \rightarrow_p \mathbb{N}_0$:
  ditto; numbers encoded in binary and separated by #