

Theory of Computer Science

D1. Turing-Computability

Gabriele Röger

University of Basel

April 8, 2019

Theory of Computer Science

April 8, 2019 — D1. Turing-Computability

D1.1 Turing-Computable Functions

D1.2 Summary

Overview: Course

contents of this course:

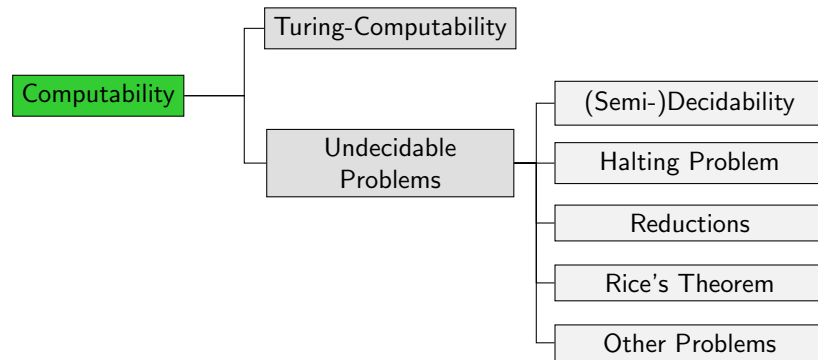
- A. background ✓
 - ▷ mathematical foundations and proof techniques
- B. logic ✓
 - ▷ How can knowledge be represented?
 - How can reasoning be automated?
- C. automata theory and formal languages ✓
 - ▷ What is a computation?
- D. Turing computability
 - ▷ What can be computed at all?
- E. complexity theory
 - ▷ What can be computed efficiently?
- F. more computability theory
 - ▷ Other models of computability

Main Question

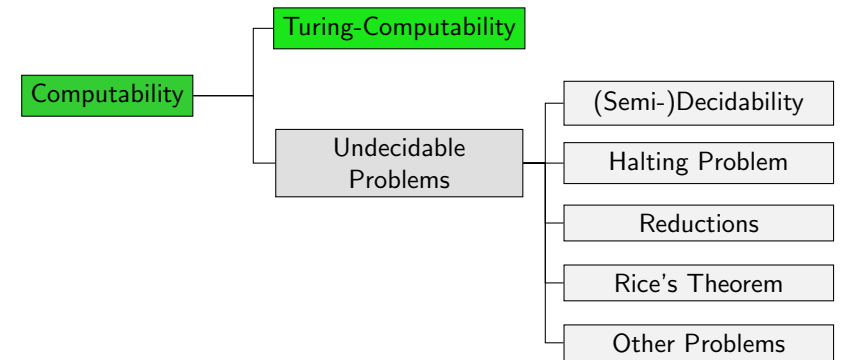
Main question in this part of the course:

**What can be computed
by a computer?**

Overview: Computability Theory



Overview: Computability Theory



D1.1 Turing-Computable Functions

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

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 (\rightsquigarrow **part F**)

Reminder: Deterministic Turing Machine (DTM)

Definition (Deterministic Turing Machine)

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

- ▶ Q finite, non-empty set of **states**
- ▶ $\Sigma \neq \emptyset$ finite **input alphabet**
- ▶ $\Gamma \supset \Sigma$ finite **tape alphabet**
- ▶ $\delta : (Q \setminus E) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, N\}$ **transition function**
- ▶ $q_0 \in Q$ **start state**
- ▶ $\square \in \Gamma \setminus \Sigma$ **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' ($c = c_0 \vdash c_1 \vdash c_2 \vdash \dots \vdash c_{n-1} \vdash c_n = c'$, $n \geq 0$)

(Definition of \vdash , i.e., how a computation step changes the configuration, is not repeated here. \rightsquigarrow **Chapter C7**)

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**: $f : \Sigma^* \rightarrow_p \Sigma^*$
- ▶ interpretation as functions on **numbers** $f : \mathbb{N}_0^k \rightarrow_p \mathbb{N}_0$: 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 $f : \Sigma^* \rightarrow_p \Sigma^*$ 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, $f : \Sigma^* \rightarrow_p \Sigma^*$)

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

German: Turing-berechenbar

Example: Turing-Computable Functions on Words

Example

Let $\Sigma = \{a, b, \#\}$.

The function $f : \Sigma^* \rightarrow_p \Sigma^*$ with $f(w) = w\#w$ for all $w \in \Sigma^*$ is Turing-computable.

↪ blackboard

Encoding Numbers as Words

Definition (Encoded Function)

Let $f : \mathbb{N}_0^k \rightarrow_p \mathbb{N}_0$ be a (partial) function.

The **encoded function** f^{code} of f is the partial function $f^{\text{code}} : \Sigma^* \rightarrow_p \Sigma^*$ 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
- ▶ $f(n_1, \dots, n_k) = n'$,
- ▶ $w = \text{bin}(n_1)\#\dots\#\text{bin}(n_k)$ and
- ▶ $w' = \text{bin}(n')$.

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

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, $f : \mathbb{N}_0^k \rightarrow_p \mathbb{N}_0$)

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

German: Turing-berechenbar

Example: Turing-Computable Numerical Function

Example

The following numerical functions are Turing-computable:

- ▶ $\text{succ} : \mathbb{N}_0 \rightarrow_p \mathbb{N}_0$ with $\text{succ}(n) := n + 1$
- ▶ $\text{pred}_1 : \mathbb{N}_0 \rightarrow_p \mathbb{N}_0$ with $\text{pred}_1(n) := \begin{cases} n - 1 & \text{if } n \geq 1 \\ 0 & \text{if } n = 0 \end{cases}$
- ▶ $\text{pred}_2 : \mathbb{N}_0 \rightarrow_p \mathbb{N}_0$ with $\text{pred}_2(n) := \begin{cases} n - 1 & \text{if } n \geq 1 \\ \text{undefined} & \text{if } n = 0 \end{cases}$

↪ blackboard/exercises

More Turing-Computable Numerical Functions

Example

The following numerical functions are Turing-computable:

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

↪ sketch?

D1.2 Summary

Summary

- ▶ main question: **what can a computer compute?**
- ▶ approach: investigate **formal models of computation**
- ▶ here: 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 #