Theory of Computer Science

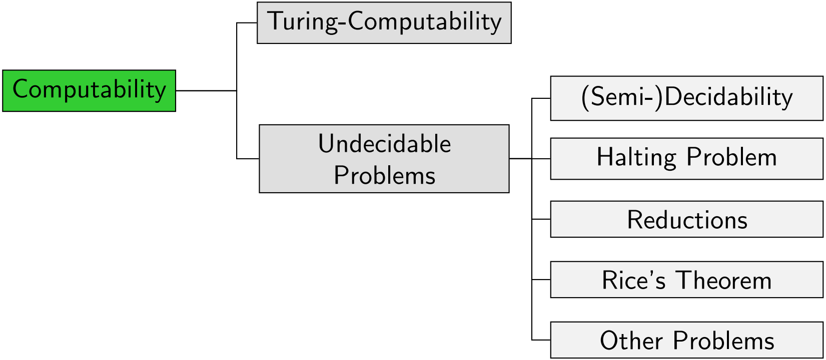
D2. Recursive Enumerability and Decidability

Gabriele Röger

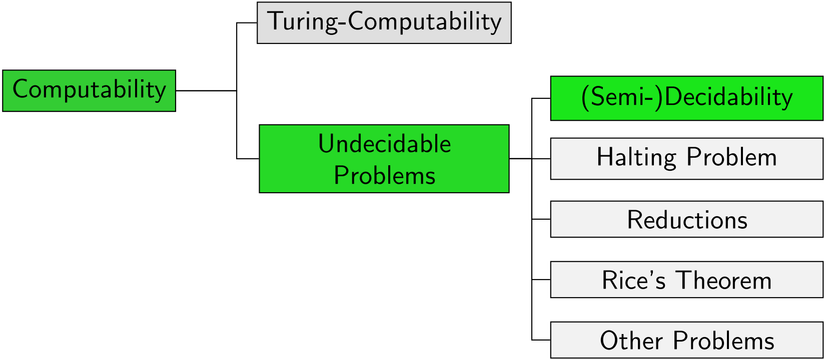
University of Basel

April 10, 2019

Overview: Computability Theory



Overview: Computability Theory



D2.1 Introduction

Guiding Question

Guiding question for next chapters:

Which kinds of problems cannot be solved by a computer?

Computable Functions

For a higher level of abstraction, we consider the Church-Turing thesis to be correct (we will further back this up in part F).

- Instead of saying Turing-computable, we just say computable.
- Instead of presenting TMs we use pseudo-code.
- Instead of only considering computable functions over words ($\Sigma^* \to_p \Sigma^*$) or numbers ($\mathbb{N}_0^k \to_p \mathbb{N}_0$), we permit arbitrary domains and codomains (e.g., $\Sigma^* \to_p \{0, 1\}$, $\mathbb{N}_0 \to \Sigma^*$), ignoring details of encoding.

Computability vs. Decidability

- last chapter: computability of functions
- now: analogous concept for languages

Why languages?

- Only yes/no questions ("Is $w \in L$?") instead of general function computation ("What is $f(w)$?") makes it easier to investigate questions.
- Results are directly transferable to the more general problem of computing arbitrary functions. ($\rightsquigarrow$ "playing 20 questions")

How do we proceed?

- We first get to know computable functions for encoding pairs of numbers as numbers (later used for dovetailing).
- Then we consider two new concepts
  - recursive enumerability and
  - semi-decidability

  and relate them to each other and earlier concepts.
- Afterwards, we require termination of algorithms
  $\rightsquigarrow$ decidability

D2.2 Encoding/Decoding Functions

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 (German: Cantorsche Paarungsfunktion)
- *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

D2.3 Recursive Enumerability

Recursive Enumerability: Definition

Definition (Recursively Enumerable)

A language $L \subseteq \Sigma^*$ is called recursively enumerable if $L = \emptyset$ or if there is a total and computable function $f : \mathbb{N}_0 \to \Sigma^*$ such that

$$L = \{f(0), f(1), f(2) \dots \}.$$

We then say that f (recursively) enumerates L.

Note: f does not have to be injective!

German: rekursiv aufzählbar, f zählt L (rekursiv) auf
$\rightsquigarrow$ do not confuse with "abzählbar" (countable)

Recursive Enumerability: Examples (1)

- $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, $f(x) = \mathtt{a}^x$ enumerates $\{\varepsilon, \mathtt{a}, \mathtt{aa}, \dots\}$.
- $\Sigma = \{\mathtt{a}, \mathtt{b}, \dots, \mathtt{z}\}$, $f(x) = \begin{cases} \mathtt{hund} & \text{if } x \bmod 3 = 0 \\ \mathtt{katze} & \text{if } x \bmod 3 = 1 \\ \mathtt{superpapagei} & \text{if } x \bmod 3 = 2 \end{cases}$

  enumerates $\{\mathtt{hund}, \mathtt{katze}, \mathtt{superpapagei}\}$.
- $\Sigma = \{0, \dots, 9\}$, $f(x) = \begin{cases} 2^x - 1 \text{ (as digits)} & \text{if } 2^x - 1 \text{ prime} \\ 3 & \text{otherwise} \end{cases}$

  enumerates Mersenne primes.

Recursive Enumerability: Examples (2)

For every alphabet Σ, the language Σ^* can be recursively enumerated with a function $f_{\Sigma^*} : \mathbb{N}_0 \rightarrow \Sigma^*$. (How?)

D2.4 Semi-Decidability

Semi-Decidability

Definition (Semi-Decidable)

A language $L \subseteq \Sigma^*$ is called semi-decidable if the following function $\chi'_L : \Sigma^* \rightarrow_{\mathrm{p}} \{0, 1\}$ is computable.

Here, for all $w \in \Sigma^*$:

$$\chi'_L(w) = \begin{cases} 1 & \text{if } w \in L \\ \textit{undefined} & \text{if } w \notin L \end{cases}$$

German: semi-entscheidbar

Type-0 Languages vs. Semi-Decidability

- Consider a DTM M that accepts a language L.
- On input w
  - M stops after a finite number of steps in an end state if $w \in L$.
  - For $w \notin L$, the computation does not terminate.
- We can easily create a DTM M' from M that computes χ'_L. (How?)
- Vice versa, given a DTM that computes χ'_L for some language L, we can derive a DTM that accepts L.

Theorem (Semi-Decidable = Type 0)

A language L is of type 0 iff L is semi-decidable.

Recursive Enumerability and Semi-Decidability (1)

Theorem (Recursively Enumerable = Semi-Decidable)

A language L is recursively enumerable iff L is semi-decidable.

Proof.

Special case $L = \emptyset$ is not a problem. (Why?)

Thus, let $L \neq \emptyset$ be a language over the alphabet Σ.

($\Rightarrow$): L is recursively enumerable.
Let f be a function that enumerates L.

Then this is a semi-decision procedure for L, given input w:

```
FOR n := 0, 1, 2, 3, ... DO
   IF f(n) = w THEN
      RETURN 1
   END
DONE
```

...

Recursive Enumerability and Semi-Decidability (2)

Proof (continued).

($\Leftarrow$): L is semi-decidable with semi-decision procedure M.
Choose $\tilde{w} \in L$ arbitrarily. (We have $L \neq \emptyset$.)

Define:

$$f(n) = \begin{cases} f_{\Sigma^*}(x) & \text{if } n \text{ is the encoding of pair } \langle x, y \rangle \\ & \text{and } M \text{ executed on } f_{\Sigma^*}(x) \text{ stops in } y \text{ steps} \\ \tilde{w} & \text{otherwise} \end{cases}$$

f is total and computable and has codomain L.
Therefore f enumerates L. □

f uses idea of dovetailing: interleaving unboundedly many computations by starting new computations dynamically forever

Characterizations of Semi-Decidability

Theorem

Let L be a language. The following statements are equivalent:

1. *L is semi-decidable.*
2. *L is recursively enumerable.*
3. *L is of type 0.*
4. *$L = \mathcal{L}(M)$ for some Turing machine M*
5. *χ'_L is (Turing-) computable.*
6. *L is the domain of a computable function.*
7. *L is the codomain of a computable function.*

Characterizations of Semi-Decidability: Proof (1)

Proof.

(1) $\Leftrightarrow$ (5): definition of semi-decidability

(1) $\Leftrightarrow$ (2): earlier theorem in this chapter

(4) $\Leftrightarrow$ (5): earlier theorem in this chapter

(3) $\Leftrightarrow$ (4): from Chapter C8

(5) $\Rightarrow$ (6): χ'_L is computable with domain L

(6) $\Rightarrow$ (5): to compute χ'_L, compute a function with domain L, then return 1

(2) $\Rightarrow$ (7): use a function enumerating L (special case $L = \emptyset$) ...

Characterizations of Semi-Decidability: Proof (2)

Proof (continued).

(7) $\Rightarrow$ (2): If $L = \emptyset$, obvious.

Otherwise, choose $\tilde{w} \in L$ arbitrarily, and let M be an algorithm computing $g : \Sigma^* \rightarrow_{\mathrm{p}} \Sigma^*$ with codomain L.

To compute a function f enumerating L,
use the same dovetailing idea as in our earlier proof:

$$f(n) = \begin{cases} g(f_{\Sigma^*}(x)) & \text{if } n \text{ is the encoding of pair } \langle x, y \rangle \\ & \text{and } M \text{ executed on } f_{\Sigma^*}(x) \text{ stops in } y \text{ steps} \\ \tilde{w} & \text{otherwise} \end{cases}$$

□

D2.5 Decidability

Semi-Decidability

Definition (Semi-Decidable)

A language $L \subseteq \Sigma^*$ is called semi-decidable if $\chi'_L : \Sigma^* \to_{\mathrm{p}} \{0, 1\}$ is computable.

Here, for all $w \in \Sigma^*$:

$$\chi'_L(w) = \begin{cases} 1 & \text{if } w \in L \\ \textit{undefined} & \text{if } w \notin L \end{cases}$$

For $w \notin L$, the computation does not (have to) terminate.

Decidability

Definition (Decidable)

A language $L \subseteq \Sigma^*$ is called decidable if $\chi_L : \Sigma^* \to \{0, 1\}$, the characteristic function of L, is computable.

Here, for all $w \in \Sigma^*$:

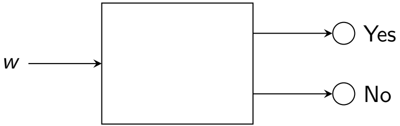
$$\chi_L(w) := \begin{cases} 1 & \text{if } w \in L \\ 0 & \text{if } w \notin L \end{cases}$$

German: entscheidbar, charakteristische Funktion

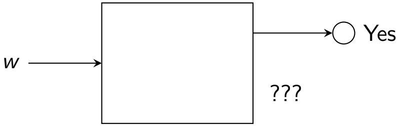
Decidability and Semi-Decidability: Intuition

Are these two definitions meaningfully different? Yes!

decidability:



semi-decidability:



Example: Diophantine equations

Connection Decidability/Semi-Decidability (1)

Theorem (Decidable vs. Semi-Decidable)

A language L is decidable iff both L and $\bar{L}$ are semi-decidable.

Proof.

($\Rightarrow$): obvious (Why?) ...

Connection Decidability/Semi-Decidability (2)

Proof (continued).

($\Leftarrow$): Let M_L be a semi-deciding algorithm for L, and let $M_{\bar{L}}$ be a semi-deciding algorithm for $\bar{L}$.

The following algorithm then is a decision procedure for L, i.e., computes $\chi_L(w)$ for a given input word w:

```
FOR s := 1, 2, 3, ... DO
  IF M_L stops on w in s steps with output 1 THEN
    RETURN 1
  END
  IF M_L̄ stops on w in s steps with output 1 THEN
    RETURN 0
  END
DONE
```

□

Example: Decidable $\neq$ Known Algorithm

Computability of χ_L does not mean we know how to compute it:

- $L = \{n \in \mathbb{N} \mid$ there are n consecutive 7s in the decimal representation of $\pi\}$.
- L is decidable.
- There are either 7-sequences of arbitrary length in π (case 1) or there is a maximal number n_0 of consecutive 7s (case 2).
  - Case 1: $\chi_L(n) = 1$ for all n
  - Case 2: $\chi_L(n) = 1$ if $n \leq n_0$, otherwise it is 0
- In both cases, the functions are computable.
- We just do not know what is the correct function.

Summary

- decidability of problems (= languages) corresponds to computability of "yes/no" functions
- semi-decidability:
  - recognizing "yes" instances in finite time
  - no answer for "no" instances
- decidability of L = semi-decidability of L and $\bar{L}$
- semi-decidability = recursive enumerability
- relationship to type-0 languages