Theory of Computer Science

D6. Decidability and Semi-Decidability

Malte Helmert

University of Basel

May 8, 2017

Theory of Computer Science

May 8, 2017 — D6. Decidability and Semi-Decidability

Overview: Computability Theory

Computability Theory

- imperative models of computation:
  - D1. Turing-Computability
  - D2. LOOP- and WHILE-Computability
  - D3. GOTO-Computability
- functional models of computation:
  - D4. Primitive Recursion and μ-Recursion
  - D5. Primitive/μ-Recursion vs. LOOP-/WHILE-Computability
- undecidable problems:
  - D6. Decidability and Semi-Decidability
  - D7. Halting Problem and Reductions
  - D8. Rice's Theorem and Other Undecidable Problems
    ~~Post's Correspondence Problem~~
    ~~Undecidable Grammar Problems~~
    ~~Gödel's Theorem and Diophantine Equations~~

Further Reading (German)

Literature for this Chapter (German)

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

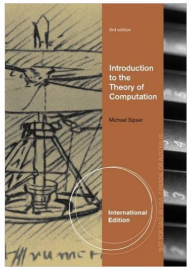
- Chapter 2.6

Further Reading (English)

Literature for this Chapter (English)

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

- Chapters 3.1 and 3.2



Notes:

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

Guiding Question

Guiding question for next three chapters:

Which kinds of problems cannot be solved by a computer?

D6.1 (Semi-) Decidability

Computable Functions

- From D1–D5, we now know enough about "computability" to use a higher level of abstraction.
- In particular, we now know that sufficiently rich computational formalisms are equivalent.

⇝ Instead of saying Turing-/WHILE-/GOTO-computable or μ-recursive, we just say computable.

⇝ Instead of presenting TMs, WHILE programs etc. in detail, 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 chapters: 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")

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

Semi-Decidability

Definition (Semi-Decidable)

A language $L \subseteq \Sigma^*$ is called semi-decidable if $\chi'_L : \Sigma^* \to_p \{0, 1\}$, "half" the characteristic function of L, is computable.

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

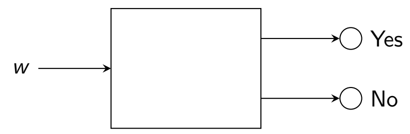
$$\chi'_L(w) = \begin{cases} 1 & \text{if } w \in L \\ \textit{undefined} & \text{if } w \notin L \end{cases}$$

German: semi-entscheidbar, "halbe" charakteristische Funktion

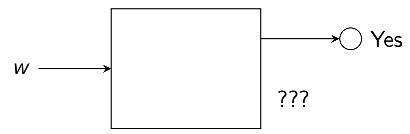
Decidability and Semi-Decidability: Intuition

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.

(⇒): obvious (Why?) ...

Connection Decidability/Semi-Decidability (2)

Proof (continued).

(⇐): 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 $s := 1, 2, 3, \dots$ DO
  IF M_L stops on w in s steps with output 1 THEN
    RETURN 1
  END
  IF $M_{\bar{L}}$ stops on w in s steps with output 1 THEN
    RETURN 0
  END
DONE

□

D6.2 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
⇝ 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 $f_{\Sigma^*} : \mathbb{N}_0 \to \Sigma^*$. (How?)

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 $L \neq \emptyset$.)

Reminder: computable encoding/decoding functions
encode, $decode_1$, $decode_2$ (Chapter D5).

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

D6.3 Models of Computation and Semi-Decidability

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-, WHILE-, GOTO-) computable. (*)*
6. *χ'_L is μ-recursive. (*)*
7. *L is the domain of a computable function.*
8. *L is the codomain of a computable function.*

(*): WHILE-/GOTO-computability and μ-recursion require encoding the input word as a number.

Characterizations of Semi-Decidability: Proof (1)

Proof.

(5) $\Leftrightarrow$ (6): equivalence of computation models (Chapters D1–D5)

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

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

(4) $\Leftrightarrow$ (5): easy to see (only distinction "acceptance" vs. "acceptance with output 1" makes no practical difference)

(3) $\Leftrightarrow$ (4): from Chapter C7

(5) $\Rightarrow$ (7): χ'_L is computable with domain L

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

(2) $\Rightarrow$ (8): use a function enumerating L (special case $L = \emptyset$) ...

Characterizations of Semi-Decidability: Proof (2)

Proof (continued).

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

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

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

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

□

D6.4 Summary

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