Theory of Computer Science

D7. Halting Problem and Reductions

Malte Helmert

University of Basel

May 10, 2017

Theory of Computer Science

May 10, 2017 — D7. Halting Problem and Reductions

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)

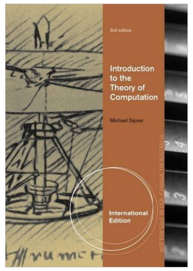
- Chapter 2.6

Further Reading (English)

Literature for this Chapter (English)

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

- Chapters 4.2 and 5.1



Notes:

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

D7.1 Introduction

Undecidable Problems

- We now know many characterizations of semi-decidability and decidability.
- What's missing is a concrete example for an undecidable (= not decidable) problem.
- Do undecidable problems even exist?
- Yes! Counting argument: there are (for a fixed Σ) as many decision algorithms (e. g., Turing machines) as numbers in $\mathbb{N}_0$ but as many languages as numbers in $\mathbb{R}$.

  Since $\mathbb{N}_0$ cannot be surjectively mapped to $\mathbb{R}$, languages with no decision algorithm exist.
- But this argument does not give us a concrete undecidable problem. $\rightsquigarrow$ main goal of this chapter

D7.2 Turing Machines as Words

Turing Machines as Inputs

- The first undecidable problems that we will get to know 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 $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \Box, E\rangle$

Idea:

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

Then it is sufficient to only encode δ explicitly:

- Q: all states mentioned in the encoding of δ
- E: all states that never occur on a left-hand side of a δ-rule
- $\Gamma = \{0, 1, \Box, a_3, a_4, \ldots, a_k\}$, 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 $D \in \{\mathrm{L}, \mathrm{R}, \mathrm{N}\}$.
- encode this rule as

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

  where $m = \begin{cases} 0 & \text{if } D = \mathrm{L} \\ 1 & \text{if } D = \mathrm{R} \\ 2 & \text{if } D = \mathrm{N} \end{cases}$
- 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)

$\delta(q_2, a_3) = \langle q_0, a_2, N \rangle$ becomes ##10#11#0#10#10
$\delta(q_1, a_1) = \langle q_3, a_0, L \rangle$ becomes ##1#1#11#0#0

Example (step 2)

##10#11#0#10#10##1#1#11#0#0
111101001101011100110100110100111101110111010111001100

Note: We can also consider the encoded word (uniquely; why?) as a number that enumerates this TM.

This is not important for the halting problem but in other contexts where we operate on numbers instead of words.

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

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

D7.3 Special Halting Problem

Special Halting Problem

Our preparations are now done and we can define:

Definition (Special Halting Problem)

The special halting problem or self-application problem is the language

$$K = \{w \in \{0,1\}^* \mid M_w \text{ started on } w \text{ terminates}\}.$$

German: spezielles Halteproblem, Selbstanwendbarkeitsproblem

Note: word w plays two roles as encoding of the TM and as input for encoded machine

Semi-Decidability of the Special Halting Problem

Theorem (Semi-Decidability of the Special Halting Problem)
The special halting problem is semi-decidable.

Proof.
We construct an "interpreter" for DTMs
that receives the encoding of a DTM as input w
and simulates its computation on input w.

If the simulated DTM stops, the interpreter returns 1.
Otherwise it does not return.

This interpreter computes χ'_K. □

Note: TMs simulating arbitrary TMs are called universal TMs.

German: universelle Turingmaschine

Undecidability of the Special Halting Problem (1)

Theorem (Undecidability of the Special Halting Problem)
The special halting problem is undecidable.

Proof.
Proof by contradiction: we assume that the special halting problem K were decidable and derive a contradiction.

So assume K is decidable. Then χ_K is computable (why?).

Let M be a Turing machine that computes χ_K, i. e.,
given a word w writes 1 or 0 onto the tape
(depending on whether $w \in K$) and then stops. ...

Undecidability of the Special Halting Problem (2)

Proof (continued).

Construct a new machine M' as follows:

1. Execute M on the input w.
2. If the tape content is 0: stop.
3. Otherwise: enter an endless loop.

Let w' be the encoding of M'. How will M' behave on input w'?

M' run on w' stops
iff M run on w' outputs 0
iff $\chi_K(w') = 0$
iff $w' \notin K$
iff $M_{w'}$ run on w' does not stop
iff M' run on w' does not stop

Contradiction! This proves the theorem. □

D7.4 Reprise: Type-0 Languages

Back to Chapter C7: Closure Properties

	Intersection	Union	Complement	Product	Star
Type 3	Yes	Yes	Yes	Yes	Yes
Type 2	No	Yes	No	Yes	Yes
Type 1	Yes(1)	Yes	Yes(1)	Yes	Yes
Type 0	Yes(1)	Yes	No(2)	Yes	Yes

Proofs?
(1) without proof
(2) proofs in later chapters (part D)

Back to Chapter C7: Decidability

	Word problem	Emptiness problem	Equivalence problem	Intersection problem
Type 3	Yes	Yes	Yes	Yes
Type 2	Yes	Yes	No	No
Type 1	Yes	No(1)	No	No
Type 0	No(2)	No(2)	No(2)	No(2)

(1) without proof
(2) proof in later chapters (part D)

Answers to Old Questions

Closure properties:

- K is semi-decidable (and thus type 0) but not decidable.

$\rightsquigarrow$ $\bar{K}$ is not semi-decidable, thus not type 0.

$\rightsquigarrow$ Type-0 languages are not closed under complement.

Decidability:

- K is type 0 but not decidable.

$\rightsquigarrow$ word problem for type-0 languages not decidable

$\rightsquigarrow$ emptiness, equivalence, intersection problem: later in exercises
(We are still missing some important results for this.)

D7.5 Reductions

What We Achieved So Far: Discussion

- We now know a concrete undecidable problem.
- But the problem is rather artificial:
  how often do we want to apply a program to itself?
- We will see that we can derive further (more useful) undecidability results from the undecidability of the special halting problem.
- The central notion for this is reducing a new problem to an already known problem.

Reductions: Definition

Definition (Reduction)

Let $A \subseteq \Sigma^*$ and $B \subseteq \Gamma^*$ be languages, and let $f : \Sigma^* \to \Gamma^*$ be a total and computable function such that for all $x \in \Sigma^*$:

$$x \in A \quad \text{if and only if} \quad f(x) \in B.$$

Then we say that A can be reduced to B (in symbols: $A \leq B$), and f is called reduction from A to B.

German: A ist auf B reduzierbar, Reduktion von A auf B

Reduction Property

Theorem (Reductions vs. Semi-Decidability/Decidability)

Let A and B be languages with $A \leq B$. Then:

1. *If B is decidable, then A is decidable.*
2. *If B is semi-decidable, then A is semi-decidable.*
3. *If A is not decidable, then B is not decidable.*
4. *If A is not semi-decidable, then B is not semi-decidable.*

$\rightsquigarrow$ In the following, we use 3. to show undecidability for further problems.

Reduction Property: Proof

Proof.

for 1.: The following algorithm computes $\chi_A(x)$ given input x:

```
$y := f(x)$
result $:= \chi_B(y)$
RETURN result
```

for 2.: identical to (1), but use χ'_B (instead of χ_B) to compute χ'_A (instead of χ_A)

for 3./4.: contrapositions of 1./2. $\rightsquigarrow$ logically equivalent □

Reductions are Preorders

Theorem (Reductions are Preorders)

The relation "$\leq$" is a preorder:

1. *For all languages A:*
   $A \leq A$ (reflexivity)
2. *For all languages A, B, C:*
   If $A \leq B$ and $B \leq C$, then $A \leq C$ (transitivity)

German: schwache Halbordnung/Quasiordnung, Reflexivität, Transitivität

Reductions are Preorders: Proof

Proof.

for 1.: The function $f(x) = x$ is a reduction from A to A because it is total and computable and $x \in A$ iff $f(x) \in A$.

for 2.: $\rightsquigarrow$ exercises $\square$

D7.6 Summary

Summary

- The special halting problem (self-application problem) is undecidable.
- However, it is semi-decidable.
- important concept in this chapter:
  Turing machines represented as words
  $\rightsquigarrow$ Turing machines taking Turing machines as their input
- reductions: "embedding" a problem as a special case of another problem
- important method for proving undecidability:
  reduce from a known undecidable problem to a new problem