Undecidable sets[*]

In these notes we present several decision problems for which no algorithm solves all instances correctly. The goal is to train to recognize such problems. If in the process of solving a more general problem, one is faced with such an intrinsically hard problem, it is a sign that the theoretical analysis needs to be changed. Maybe one should make an addtional assumption on the instances for which the problem needs to be solved? Perhaps then, one can obtain a solution that provably works. And maybe it is also useful in practice?

At the moment, we still study computability theory, and we do not worry about time and space efficiency of our algorithms. We do wonder whether there exists an algorithm that halts on all instances and only produces good answers. Computability theory is interesting for its negative results: if there is no algorithm at all, there is definitely no practical algorithm. We introduce techniques such as diagonalization and reduction. These techniques are also used in the part about computational complexity.

1 Undecidable sets exist

The diagonalization argument is an important technique to show that two sets are different. Historically, it was invented to show that there exist two types of infinite sets: countable and uncountable. We first explain this result. Afterwards we show that the halting problem for Turing machines is undecidable.

1.1 Countable sets

A set is *countable* if there exists a finite or infinite list that contains all the elements of the set (in any order, possibly with repetitions). By definition, finite sets are countable. The natural numbers are also countable: they all appear in the list $1, 2, 3, 4, \dots$ Many other sets are countable too:

Integer numbers	0	-1	1	-2	2	-3	3	$\dots$
Pairs of natural numbers	$(1,1)$	$(1,2)$	$(2,1)$	$(3,1)$	$(2,2)$	$(1,3)$	$\dots$	
Positive rational numbers	$1/1$	$1/2$	$2/1$	$3/1$	$2/2$	$1/3$	$\dots$	
Bitstrings	ε	0	1	00	10	01	11	$\dots$

Figure 1: Countable sets

[*]Theoretical Computer Science 2016-2017, National Research University Higher School of Economics, First version: Februari 18th 2018
Update Februari 28th 2018: moved solutions to the end, language corrections, corrected typos in exercise 3.
Update May 21st 2018: minor textual improvements, statement of Theorem 12, and a rewrite of Section 2.

It is easy to believe that every bitstring x appears in the list of bitstrings above. Even more, the position of a string $x = x_1 \dots x_n$ in this list can be determined explicitly: $2^n + \sum_{i=1}^{n} x_i 2^{i-1}$.

A subset of a countable set A is also countable, because we can simply drop the elements in the list of A that do not belong to the subset. All sets whose elements can be encoded as bitstrings, are countable, because the set of bitstrings is countable. (Vice versa, to all countable sets, we can assign bitstrings.) There are countably many Turing machines, and hence, the class of decidable sets is countable.

We give some more ways to observe that a set is countably infinite. Assume that A and B are countable.

- The union of two countable sets is also countable. Indeed, if $A = \{a_1, a_2, \dots\}$ and $B = \{b_1, b_2, \dots\}$ then $A \cup B = \{a_1, b_1, a_2, b_2, \dots\}$.
- The direct product of $A \times B$ is countable. We use the list for pairs of natural numbers:
$$A \times B = \{(a_1, b_1), (a_2, b_1), (a_1, b_2), (a_3, b_1), (a_2, b_2), (a_1, b_3), \dots\}.$$
- This also implies that A^k is countable for all k, using a natural correspondence between A^k and $A \times A^{k-1}$.

Exercise 1. If $A_1, A_2, \cdots$ is a sequence of countable sets, show that $A_1 \cup A_2 \cup \cdots$ is also countable. Conclude that if A is countable then also A^* is countable.

1.2 Diagonalization

We now show that not all sets are countable.

Theorem 1. *The set of infinite bitsequences is not countable.*

$$\begin{array}{r|cccccc}
1 & \mathbf{1} & 0 & 1 & 1 & 1 & \dots \\
2 & 0 & \mathbf{1} & 0 & 0 & 1 & \dots \\
3 & 0 & 1 & \mathbf{1} & 1 & 0 & \dots \\
4 & 0 & 0 & 0 & \mathbf{0} & 1 & \dots \\
5 & 0 & 1 & 1 & 0 & \mathbf{0} & \dots \\
\vdots & & & \vdots & & & \\
\hline
\text{inverse} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \dots \\
\text{diagonal} & & & & & &
\end{array}$$

Figure 2: The method of diagonalization

Proof. We need to show that for every list of sequences, there exists a sequence that does not appear in this list. A list of sequences specifies a table as illustrated in figure 2. Consider the *inverted diagonal*, it is, the sequence containing the inverse of the 1st bit of the 1st sequence, the inverse of the 2nd bit of the 2nd sequence, etc., see figure 2. Suppose that this sequence is in the list. Let i be the index of its row. By construction, the ith bit must be equal to its inverse, and this is impossible. Hence, this sequence does not belong to the list. The theorem is proven. □

The same idea is also illustrated by a funny story for children. Imagine that in some street there is a barber shop and the following rule holds:

The barber shaves everyone who lives in the street and does not shave himself.

Does the barber also live in the street or does he only work there and lives elsewhere? Assume he lives in the street. Now we ask ourself the following question: Does the barber shave himself? If he shaves himself, he should not shave himself. If he does not shave himself, he should shave himself. A contradiction. Our assumption must be false: the barber does not live in the street.

From Theorem 1 we can also conclude that the real numbers in the closed-open interval $[0,1)$ are uncountable. To every real number in $[0,1)$, we can associate at most 2 binary representations, i.e., two infinite bitsequences. Note that some reals have 2 binary representations, for example $0.0111\cdots = 0.1000\ldots$ Hence, if the reals were countable, then we could also obtain a list of infinite bitsequences.

With each subset $S \subseteq \mathbb{N}$ we can associate a bitsequence $s = s_1 s_2 \ldots$ where $s_n = 1$ if $n \in S$ and $s_n = 0$ otherwise. Each sequence specifies precisely one such set. Hence, the number of subsets of $S \subseteq \mathbb{N}$ is also uncountable. Because there are countably many Turing machines we immediately obtain the following result.

Theorem 2. *There exists an undecidable set $S \subseteq \mathbb{N}$.*

Exercise 2. Consider the set of bitsequences with finitely many 1's, i.e., sets of sequences $a_1 a_2 a_3 \ldots$ such that $\sum_{i=0}^{\infty} a_i$ is finite. Is this set countable or uncountable?

In fact, "most" sets are undecidable.

Theorem 3. *Generate an infinite bitsequence $a_1 a_2 a_3 \ldots$ by choosing all $a_i \in \{0,1\}$ independently and randomly. Then, the set $\{n\colon a_n = 1\}$ is undecidable with probability 1.*

Proof. We need to show that for each $\ell \in \mathbb{N}$, the probability that the set is computable is at most $2^{-\ell}$. Let $s_1, s_2, \ldots$ be a list of all computable infinite sequences.

$$\begin{aligned} \Pr\left\{a_1 \ldots a_{\ell+1} = s_{1,1} \ldots s_{1,\ell+1}\right\} &= 2^{-\ell-1} \\ \Pr\left\{a_2 \ldots a_{\ell+2} = s_{2,1} \ldots s_{2,\ell+1}\right\} &= 2^{-\ell-2} \\ &\cdots \end{aligned}$$

By the union bound, the probability that $a_1 a_2 \ldots$ equals one of the sequences $s_1, s_2, \ldots$ is at most $2^{-\ell-1} + 2^{-\ell-2} + \ldots = 2^{-\ell}$. $\square$

Exercise 3. Recall that computable functions are by definition total. Let $\varphi\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be a computable function. Show that there exists a computable function f such that for all n: $f(\cdot) \neq \varphi(n, \cdot)$.

Exercise 4. Imagine there exists a programming language and a code checker. This checker inspects some code in this language and verifies whether it computes a total one-argument function from $\mathbb{N}$ to $\mathbb{N}$. Imagine this verifier is computable and correct for all possible codes. In other words, it decides the language of all programs that implement total functions. Use the exercise 3 to argue that there exists a computable function that can not be implemented in this language. Hint: generate a list of all programs $p_1, p_2, \ldots$ that pass the code checker.

1.3 The Halting set is undecidable

For a Turing machine M, let $\langle M \rangle$ represent a description of the machine of M.

Theorem 4. *The following set is undecidable*

$$\mathrm{H}_{\mathrm{TM}} = \{\langle M, w\rangle \colon M \textit{ halts on input } w\}$$

A funny animation of the proof is here:
`http://www.youtube.com/watch?v=92WHN-pAFCs&t=100`

Proof. Consider a list of *all* Turing machines $M_1, M_2, \ldots$ and the following table that indicates whether on some input w, machine M_i halts (`h`) or runs forever (∞).

	ε	0	1	00	$\ldots$
M_1	$\underline{\texttt{h}}$	∞	∞	`h`	$\ldots$
M_2	`h`	$\underline{\infty}$	`h`	`h`	$\ldots$
M_3	∞	∞	$\underline{\texttt{h}}$	`h`	$\ldots$
M_4	∞	`h`	`h`	$\underline{\infty}$	$\ldots$
$\vdots$			$\vdots$		
inverse diagonal	$\underline{\infty}$	$\underline{\texttt{h}}$	$\underline{\infty}$	$\underline{\texttt{h}}$	$\ldots$

We define the inverse diagonal as usual. By definition, this inverse diagonal can not be a row of the table. Now assume that the table is decidable. Then one can construct a machine that behaves like the inverse diagonal: on input a string w, it computes the value of the corresponding cell in the diagonal, and if this value is `h`, it goes in a loop, otherwise it halts. Thus, by assumption, the inverse diagonal describes some machine, and therefore, this row must appear in the table. But by construction the row can not appear in the table, a contradiction. Hence our assumption must be false: the table can not be decidable. $\square$

Exercise 5. An extension of a partial function g is a partial function which equals g on all arguments for which g is defined. (But might also be defined elsewhere.) Show that there exists a binary partial computable function $g \colon \{0,1\}^* \to \{0,1\}$ that has no computable extension. Hint: choose g such that $g(\langle M\rangle) = 1 - M(\langle M\rangle)$ if $M(\langle M\rangle) \in \{0,1\}$.

2 Recognizable sets

The halting set is not decidable, but its elements are recognizable: if an element belongs to the set, we can verify this by running the machine on the input. On the other hand, there might not be a way to verify that an element is outside the set.

Definition 5. *A language L over Σ is* recognizable[1] *if there exists a Turing machine that on input any string $w \in \Sigma^*$, halts if and only if $w \in L$.*

[1] This term was defined in Sipser's book using a slightly different characterization, but it is easily seen to be equivalent.

The halting set $\mathrm{H_{TM}}$ is recognizable: consider the algorithm that on input $\langle M, w\rangle$ simulates M on input w and halts if the simulation halts.

An equivalent way to define the class of recognizable languages, is to use Turing machines with a special *one-way* output tape. On this tape, the machine can only move in one direction, and hence it can not modify a cell after it has written to it and moved away. On this tape, it can write strings over some alphabet, separated by a blank. (We assume the blank does not belong to the input alphabet.) The machine is allowed to compute forever, and on its output tape it can generate an infinite stream of strings. In this way, the machine *enumerates* a set.

Definition 6. *A set is* enumerable *if there exists a Turing machine that has an additional one-way output tape, on which it prints a sequence of elements such that this sequence and the set contain precisely the same elements.*

Definitions 5 and 6 are equivalent.

Lemma 7. *A language is enumerable if and only if it is recognizable.*

Proof. Enumerable ⇒ recognizable. Consider the following algorithm: on input x, enumerate the stream and wait until x appears. If this happens, halt. Otherwise, let the simulation run forever.

Recognizable ⇒ enumerable. Let M be a machine that halts on precisely the elements of the language. The algorithm that enumerates the stream searches for all inputs on which M produces some output. This can be done for example as follows. For $t = 1, 2, \ldots$, it evaluates M on all inputs of length at most t during t steps; then it writes all inputs for which M has halted on the output tape. □

Recall that the decidable sets $S \subseteq \mathbb{N}$ are those sets that are the image of a non-decreasing computable function. (Recall our convention that computable functions are total.) Also recognizable languages can be characterized using computable functions.

Exercise 6. Show that a nonempty set $A \subseteq \Sigma^*$ is enumerable if and only if it is the image of a computable function $f \colon \mathbb{N} \to \Sigma^*$.

Exercise 7. Show that a set $A \subseteq \Sigma^*$ is enumerable if and only if there exists a computable predicate f (i.e., a function $P \colon \mathbb{N} \times \Sigma^* \to \{\texttt{True, False}\}$), such that $x \in A \Leftrightarrow \exists n \colon P(n, x)$.

Exercise 8. Show that:

1. The union and intersection of two recognizable sets are recognizable.

2. The join $A \oplus B$ of two sets in $\{0,1\}^*$ is given by $\{0x : x \in A\} \cup \{1x : x \in B\}$. The join of two sets is recognizable if and only if both sets are recognizable.

3. A language is decidable if and only it is recognizable and its complement is recognizable. Thus the complement of $\mathrm{H_{TM}}$ is not recognizable.

4. There exists a language that is not recognizable for which the complement is also not recognizable.

5. Adapt the proof of Theorem 3 to show that for most binary functions f, the set $\{n : f(n) = 1\}$ is not recognizable.

3 Mapping reducibility

The English word *to reduce* has several meanings. According to the Oxford dictionary, the first meaning is: "to make smaller or less in size, amount, or degree". Another meaning is: "to transform to a different or more basic form". We use it in the latter meaning. As an introduction, we quote page 171 from Sipser's book.

> A *reduction* is a way of converting one problem to another problem such that a solution for the second problem can be used to solve the first problem.
>
> Such reductions come up in everyday life, even if we don't usually refer to them this way. For example, suppose that you want to find your way around in a new city. You know that this would be easy if you had a map. Thus you can reduce the problem of finding your way around to the problem of obtaining a map of the city.
>
> Reducibility always involves two problems, which we call A and B. If A reduces to B, we can use a solution for B to solve A. So in our example, A is the problem of finding your way around the city, and B is the problem of obtaining a map.

Below we focus on decision problems of languages.

It is interesting to note that the word "reduce" originates from the Latin word *ducere*, which means 'to lead' or 'to guide'. This word also appears in many other English words: to introduce, to induce, induction (a proof technique), to deduce, to conduct, conductor (of an orchestra or an electrical conductor), to produce, to seduce, to educate.[2] 'Abduction' means 'kidnapping' and a 'viaduct' is a long bridge. Maybe you have heard about 'transductive learning' in machine learning?

3.1 Definition

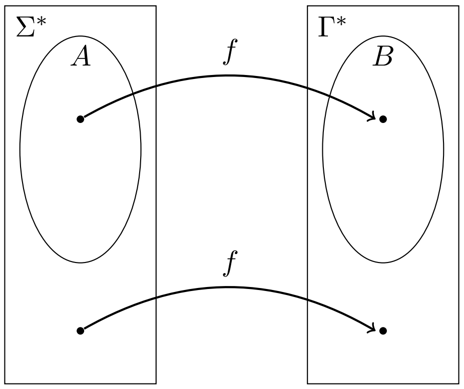
Definition 8. *A language A over an alphabet Σ* mapping reduces[3] *to a language B over an alphabet Γ, notation $A \leq_m B$ if there exists a computable function $f : \Sigma^* \to \Gamma^*$ such that for all $w \in \Sigma^*$*

$$w \in A \iff f(w) \in B.$$

The function f is called the reduction.

[2] *To educate* means: to give intellectual, moral, and social instruction to someone, typically at a school or university. It comes from *ducare*, a stronger form of *ducere*, that probably means 'to guide repetitively'. The prefix 'e-' or 'ex-' means 'outwards' or 'upwards'. Romans used the word *educare* in the meaning of 'to teach a child to behave well', literally: to repetitively guide someone until he is ready to leave the house. The idea that moral and social instruction should (also) happen in schools appeared much later, in the 16th century. Thanks to Sam Creve, a Latin teacher.

[3] This name was invented by M. Sipser because it better describes the definition. The standard name is *many-one reduction.*



If A reduces to B, then given a solution of a decision problem for B, we can solve the decision problem for A.

Lemma 9. *Assume $A \leq_m B$.*
1. If B is decidable, then A is decidable.
2. If B is recognizable, then A is recognizable.
3. If A is not decidable, then B is not decidable.
4. If A is not recognizable, then B is not recognizable.

Proof. Let f be the function in the reduction from A to B.

1. By assumption there exists a decision algorithm for B. The following algorithm decides A: on input x it calculates $f(x)$ and runs the decision algorithm for B.

2. Fix a partial function g whose domain equals B. The partial function $x \mapsto g(f(x))$ is partial computable and is defined precisely on the elements of A.

Items 3 and 4 are the contrapositive of items 1 and 2. □

Unfortunately, the first meaning of the English word "reduce" is rather confusing. When we reduce A to B, this does not imply that B is easier than A. In fact the opposite: B might be harder to decide than A. This is also why the notation $A \leq_m B$ is useful.

Lemma 10. *If $A \leq_m B$ and $B \leq_m C$ then $A \leq_m C$.*

Proof. A composition of computable functions is computable as well. Hence, if f is a reduction from A to B, and g is a reduction from B to C, then $x \mapsto g(f(x))$ is a reduction from A to C:

$$x \in A \quad \iff \quad f(x) \in B \quad \iff \quad g(f(x)) \in C. \qquad \square$$

Lemma 10 and the 3th item of Lemma 9 provide a method to show that a set X is undecidable: we find sets A, B, ..., E such that

$$\mathrm{H_{TM}} \leq_m A \leq_m B \leq_m \ldots \leq_m E \leq_m X.$$

3.2 Examples

We define the recognizable set $\mathrm{H}_{\mathrm{TM},\varepsilon}$ and repeat the definition of $\mathrm{H_{TM}}$:

$$\begin{aligned} \mathrm{H}_{\mathrm{TM},\varepsilon} &= \{\langle M\rangle \colon M \text{ is a TM that halts on input the empty string}\} \\ \mathrm{H_{TM}} &= \{\langle M, w\rangle \colon M \text{ is a TM that halts on input } w\}. \end{aligned}$$

We show that $\mathrm{H}_{\mathrm{TM},\varepsilon} \leq_m \mathrm{H}_{\mathrm{TM}}$.

Note that $\mathrm{H}_{\mathrm{TM},\varepsilon}$ is a special case of the problem H_{TM}. Hence, the reduction f is simple: f maps every description $\langle M\rangle$ to a description $\langle M, \varepsilon\rangle$, and every invalid description of a machine to an invalid description. Such a transformation is computable, and almost by definition:

$$x \in \mathrm{H}_{\mathrm{TM},\varepsilon} \quad \Longleftrightarrow \quad f(x) \in \mathrm{H}_{\mathrm{TM}}.$$

Indeed, if $\langle M\rangle \in \mathrm{H}_{\mathrm{TM},\varepsilon}$, then $\langle M, \varepsilon\rangle \in \mathrm{H}_{\mathrm{TM}}$. If $x \notin \mathrm{H}_{\mathrm{TM},\varepsilon}$, then either x is not a description of a machine, thus $f(x)$ is also an invalid description, or x represents a machine $\langle M\rangle$ that does not halt on empty input. Thus $\langle M, \varepsilon\rangle \notin \mathrm{H}_{\mathrm{TM}}$ and f is the required reduction.

Unfortunately, our previous result does not help to prove that $\mathrm{H}_{\mathrm{TM},\varepsilon}$ is undecidable. To apply Lemma 9, we need the reduction in the other direction. We now prove that $\mathrm{H}_{\mathrm{TM}} \leq_m \mathrm{H}_{\mathrm{TM},\varepsilon}$.

For this, we map $\langle M, w\rangle$ to a machine that has the argument w hardcoded in it. In detail: let $U_{M,w}$ be the machine that on empty input writes w on the tape, returns, then runs M, and halts when M halts.

Let f be a computable function that maps $\langle M, w\rangle$ to $\langle U_{M,w}\rangle$ and invalid descriptions to invalid descriptions. By construction we have

$$M(w) \text{ halts} \quad \Longleftrightarrow \quad U_{M,w}(\varepsilon) \text{ halts}.$$

Thus, by definition

$$\langle M, w\rangle \in \mathrm{H}_{\mathrm{TM}} \quad \Longleftrightarrow \quad \langle U_{M,w}\rangle \in \mathrm{H}_{\mathrm{TM},\varepsilon}.$$

We have proven the following lemma.

Lemma 11. $\mathrm{H}_{\mathrm{TM},\varepsilon}$ *is undecidable.*

Exercise 9. Consider the set of all descriptions $\langle M\rangle$ of machines M that on input the string 000 halt with output the string 111. Adapt the proof above to argue that also this set is not computable.

We give one more example. We show that the set

$$\mathrm{E}_{\mathrm{TM}} = \big\{\langle M\rangle \colon \forall x\, [M(x) \text{ does not halt}]\,\big\}.$$

is undecidable. Note that a set is decidable if and only if its complement is decidable. We reduce the complement of $\mathrm{H}_{\mathrm{TM},\varepsilon}$ to E_{TM}. This is easy: we map a machine M to a machine U_M that runs M on empty input and halts if M halts. Note that U_M should first delete its input string before the simulation starts. The reduction is given by a computable function that maps all $\langle M\rangle$ to $\langle U_M\rangle$, and maps invalid descriptions to invalid descriptions. We have that

$$\langle M\rangle \notin \mathrm{E}_{\mathrm{TM}} \quad \Longleftrightarrow \quad \langle U_M\rangle \in \mathrm{H}_{\mathrm{TM},\varepsilon}.$$

Hence, E_{TM} is undecidable.

3.3 Exercises

Exercise 10. Show that if B is decidable, then for every nonempty language C with nonempty complement, we have that $B \leq_m C$.

Exercise 11. Show that there exists a language B such that ${}^{\mathsf{c}}B \not\leq_m B$, where ${}^{\mathsf{c}}B$ is the complement of B. Hint: use exercise 8.3.

Exercise 12. Let Σ be an alphabet. Show that there exists no set $X \subseteq \Sigma^*$ such that $B \leq_m X$ for every set $B \subseteq \Sigma^*$. Hint: use that there are uncountably many subsets of Σ^*.

3.4 Rice's theorem

The examples of subsection section 3.2 are in some sense similar. For a non-trivial property of a function, they each claim that given a description of a Turing machine, we can not tell whether the machine computes a function that has this property.

More general, Rice's theorem states that for each non-trivial property of input-output relations, and for each method that inspects Turing machines, there are always machines for which the method is unable to decide whether the machine satisfies this property.

With a *trivial* property we mean that either no Turing machine has it or all machines have it.

Theorem 12 (Rice)**.** *Let F be a set of partial functions with 1 argument.*

$$I = \{\langle M \rangle : M \textit{ is a Turing machine that computes a function in } F\}$$

is either empty, equals the set of all descriptions of Turing machines, or is undecidable.

From this we directly conclude that $\mathrm{H}_{\mathrm{TM},\varepsilon}$ and E_{TM} are undecidable: for the first, we choose F to be the set of partial functions that are undefined on the empty string, and for the second, $F = \{\text{the-everywhere-undefined-function}\}$.

Corollary 13. *For every Turing machine M there exist infinitely many other Turing machines that compute the same one argument function as M.*

Proof. Assume M computes the partial function f. Apply Theorem 12 to the set $F = \{f\}$. I can neither be empty nor contain all machines. Hence, I must be undecidable. Because all finite sets are decidable, I must be infinite and hence, infinitely many machines compute f. □

Proof of Theorem 12. First assume that the-everywhere-undefined-function does not belong to F. If F contains no partial computable functions, then I is empty and nothing needs to be proven. Otherwise, let h be a partial computable function in F. Let $P_{M,w}$ be a Turing machine that evaluates a function

$$x \mapsto \begin{cases} h(x) & \text{if } M \text{ halts on input } w, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

There is a computable function f that maps $\langle M, w \rangle$ to $\langle P_{M,w} \rangle$. f is a reduction from H_{TM} to I. Hence, I is undecidable.

The case where the everywhere undefined function does belong to F is proven in a similar way. If F contains all partial computable functions, then nothing needs to be shown. Otherwise, we choose h to be outside F. Let the function f be defined in the same way. Now f is a reduction from H_{TM} to the complement of I. Thus this complement and I are undecidable. □

Observe that the statement of the theorem is symmetric: after replacing F by its complement, we obtain a theorem that is equivalent. On the other hand, the proof above has two cases and in each case this asymmetry has disappeared. This is unsatisfactory. Below we give a proof that preserves this symmetry.

Second proof. In exercise 5 it is shown that there exists a partial computable binary function g without computable extension. Assume that there are partial computable functions h_{in} in F and h_{out} outside F. If at least one of these functions does not exist, then nothing needs to be shown. Let P_r be a machine that on input x

$$\begin{cases} \text{outputs } h_{\text{in}}(x) & \text{if } g(r) = 1 \\ \text{outputs } h_{\text{out}}(x) & \text{if } g(r) = 0 \\ \text{is undefined} & \text{otherwise.} \end{cases}$$

We show that if I were decidable, then we define the following computable extension $\tilde{g}$ of g:

$$\tilde{g}(r) = \begin{cases} 1 & \text{if } \langle P_r \rangle \in I \\ 0 & \text{if } \langle P_r \rangle \notin I. \end{cases}$$

Observe that $\tilde{g}$ is indeed computable and extends g. On the other hand, by choice of g, such a computable function can not exist. A contradiction. The theorem is proven. □

4 Undecidable problems

4.1 Integer polynomials with integer roots

The equation $x^3 + y^3 + z^3 = 29$ has an integer solution: $x = 3$ and $y = z = 1$. The equation $x^3 + y^3 + z^3 = 30$ also has solutions, but the only solutions are very large: one of the variables must exceed 10^8. Whether

$$x^3 + y^3 + z^3 = 33$$

has an integer solution is still an open problem, that has puzzled many people.

Input:	A polynomial P with integer coefficients over several variables.
Question:	Does P have an integer solution, i.e., $\exists x_1, \ldots, x_m \in \mathbb{Z}\, [P(x_1, \ldots, x_m) = 0]$?

In 1900, David Hilbert published 23 mathematical problems for the 20th century. The 10th problem asks whether there exists a finite procedure to know the answer this question. Building on work of others[4], the Russian mathematician Y. Matiyasevich's showed that the problem above is undecidable. The proof is too hard for our lectures.

It is known that the problem above is also undecidable for polynomials with 11 unknowns[5]. The problem is decidable for polynomials in 1 variable. Whether it is also decidable for polynomials in 2 variables, is unknown. There exists an algorithm that seems to be always correct. Unfortunately, the algorithm is only known to be provably correct using the abc-conjecture; this conjecture is generally believed to be true, and is perhaps one of the most important open problems in number theory.

[4] In particular M. Davis, Y. Matiyasevich, H. Putnam and J. Robinson.

[5] J.P. Jones, "Universal Diophantine equation", *Journal of Symbolic Logic* (47), (1982), 549-571.
For more details, see
Poonen, Bjorn. "UndecidabilityinNumber Theory." Notices of the AMS 55.3 (2008),
`https://mathoverflow.net/questions/51987/which-types-of-diophantine-equations-are-solvable` and
`https://mathoverflow.net/questions/207482/algorithmic-un-solvability-of-diophantine-equations-of-given-degree-with-given`.

It is not known if the following problem is decidable.

Input:	A polynomial P with integer coefficients over several variables.
Question:	Does P have a *rational* solution, i.e., $\exists x_1, \ldots, x_m \in \mathbb{Q}\,[P(x_1, \ldots, x_m) = 0]$?

In fact, it does not matter whether we consider polynomials with rational or integer coefficients: after multiplying by the product of all denominators, we obtain an equivalent integer polynomial.

4.2 Machines with finitely many registers

In the previous notes we defined register machines. Consider the halting problem for such machines.

Input:	A register machine.
Question:	Does the register machine halts.

We also showed that register machines can simulate Turing machines, hence, the problem above reduces to the halting problem on Turing machines and is hence undecidable. Formally, let

$$\mathrm{H}_{\mathrm{Reg}} = \{\langle R\rangle \colon R \text{ is a register machine that halts}\}\,.$$

The transformation of the proof implies that $\mathrm{H}_{\mathrm{TM},\varepsilon} \leq_m \mathrm{H}_{\mathrm{Reg}}$.

In one of the exercises we also showed that we can rewrite every program for a register machine using only the following 3 instructions:

- $a := a + 1$
- `If` $a = 0$ `then goto line` m `else` $a := a - 1$ `and goto line` n
- `Stop`

This will be convenient in the study of FRACTRAN and tag-systems.

4.3 FRACTRAN

The following programming language is called FRACTRAN and was invented by John Conway.

Every program consists of a number n and a list of positive rational numbers $f_1, f_2, \ldots, f_e$. The program is executed in steps. At each step, n is replaced by the left most value of $f_1 n, f_2 n, \ldots, f_e n$ that is integer. If none of these numbers are integer, the computation halts.

For example, the program given by $n = 5$ and the fractions $[2/3, 6/5, 1/2]$ halts and the subsequent values of n are: 5, 6, 4, 2, 1. Consider the halting problem for FRACTRAN programs.

Input:	A FRACTRAN program given by a number n and a list of fractions.
Question:	Does the program halt?

Exercise 13. Show that the set of halting FRACTRAN programs is undecidable.

Tip: Encode the variables $a, b, c, \ldots, g$ of a register machine as $2^a 3^b 5^c \ldots p^g$. Associate with each line number ℓ, a prime number q_ℓ different from the prime numbers used to store $a, b, c, \ldots, g$. In each computation step, let n be $q_\ell 2^a 3^b 5^c \ldots p^g$.

4.4 Tag systems

Another simple mechanism capable of computing is a k-tag system.

For each $k \geq 1$, a *k-tag system* has as input, a word over an alphabet Σ. Its operation is described by a partial function $d\colon \Sigma \to \Sigma^*$ which transforms the input string w step by step. If at some point, w has length less than k or $d(w_1)$ is undefined for the first letter w_1 of w, the computation halts (and w is the output). Otherwise, the first k letters of w are deleted, $d(w_1)$ is appended at the end of the word and the process is iterated.

For example, consider a 2-tag system over $\{0,1\}$ with $d(0) = 011$ and $d(1) = 1$. On input 01111 we obtain:

→ ~~01~~111**011** → ~~11~~1011**1** → ~~10~~111**1** → ~~11~~11**1** → ~~11~~1**1** → ~~11~~**1**.

Input:	A k-tag system over some Σ and a string $w \in \Sigma^*$.
Question:	Does the tag system halt on input w?

Exercise 14. Show that the halting problem for 1-tag systems is decidable.

Theorem 14. *The halting problem for 2-tag systems is undecidable.*

Tag systems are very simple objects to define. The smallest known universal Turing machines were designed by simulating tag systems. Moreover, they can simulate t steps of any Turing machines using a number of computation steps at most polynomial in t. 2-tag systems with a 1 or 2-symbol alphabet are easy to analyse and the Halting problem on such machines is decidable. However, for a 3-symbol alphabet there is little hope to solve the Halting problem.

To explain why, we define the Collatz conjecture. For each n, consider this sequence that is computed by iterative application of the function

$$f(n) = \begin{cases} 3n+1 & \text{if } n \text{ is odd} \\ n/2 & \text{if } n \text{ is even.} \end{cases}$$

The famous Collatz conjecture states that for every n the sequence

$$n, f(n), f(f(n)), f(f(f(n))), \ldots$$

contains a 1. For example, for $n = 7$ we obtain

$$7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1.$$

This is an other notorious difficult open question in mathematics. Maybe the set of n for which this sequence contains a 1 is decidable? Again, this is an open question.

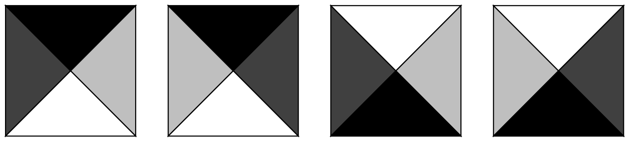
The following 2-tag system over $\Sigma = \{a, b, c\}$ halts on input a^n if and only if the sequence $n, f(n), f(f(n)), \ldots$ contains a 1:

$$\begin{aligned} a &\mapsto bc \\ b &\mapsto a \\ c &\mapsto ccc. \end{aligned}$$

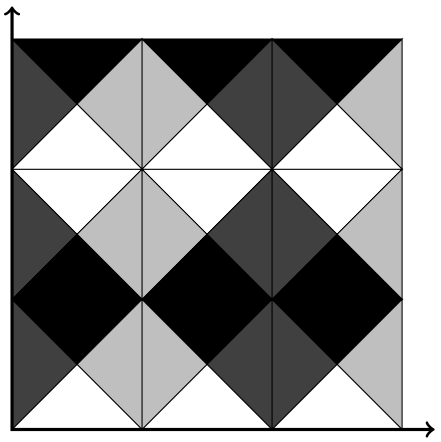
Hence, even for this tag system, there seems to be little hope to solve the Halting problem. This also explains why it is unknown whether the halting problem for many small Turing machines is undecidable.

4.5 Tilings

A Wang tile is a square characterized by 4 colors on each side. Here are some examples:



Tiles can not be rotated, only translated. With these tiles, we can fill a quarter of a plane such that touching edges have the same color:



Definition 15. *Let C be a finite nonempty set called* colors. *A* tile *t is a 4-tuple of colors. The colors of the tile are conveniently written as* $\mathrm{N}, \mathrm{E}, \mathrm{S}, \mathrm{W}$. *A* tile set *$T$ is a subset of C^4.*

A (T, t)-tiling *is a mapping $f\colon \mathbb{N} \times \mathbb{N} \to T$ with $f(1, 1) = t$ such that the colours of neighbouring tiles match, i.e., for all $(i, j) \in \mathbb{N} \times \mathbb{N}$, $f(i, j)_\mathrm{E} = f(i + 1, j)_\mathrm{W}$ and $f(i, j)_\mathrm{N} = f(i, j + 1)_\mathrm{S}$.*

Theorem 16. *The set of pairs (T, t) for which no (T, t)-tiling exists is undecidable.*

We prove this by reducing $\mathrm{H}_{\mathrm{TM},\varepsilon}$ to a set of pairs (T, t) for which no tiling exists. For each machine M, we construct a tile set T and a tile t such that there exists at most one (T, t)-tiling, and if a (T, t) exists, then it must be the computation history of a Turing machine. If the computation does not halt on empty input, the tiling fills the whole plane. Otherwise, we let a tile appear that can not be matched by any other tile.

The configurations of the Turing machine at subsequent computation steps are represented by increasing rows in the tiling. Every tile of a row encodes precisely one tape cell. In every row there are two tiles that represents the computation head and a cell. The first represents the head and its cell at the beginning of the computation step. The second tile corresponds to the end of the computation step. Obviously, these two tiles are next to each other in a tiling, see figure 3.

To initialize the computation correctly, we use some additional tiles. These tiles only appear in the first row of a tiling.

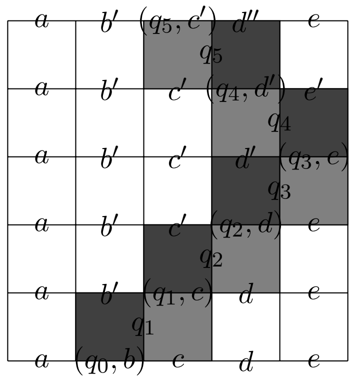

a b' (q_5, c') d'' e
q_5
a b' c' (q_4, d') e'
q_4
a b' c' d' (q_3, e)
q_3
a b' c' (q_2, d) e
q_2
a b' (q_1, c) d e
q_1
a (q_0, b) c d e

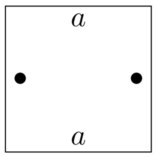
Figure 3: A tiling that simulates a Turing machine going through states $q_0, \ldots, q_5$. The machine moves three times right and one times left. Dark gray and light gray tiles represent the computation head at the beginning and end of the computation step.

Proof of Theorem 16. We consider Turing machines that make at least 1 move. Machines that immediately halt are mapped to some fixed tiling problem that can not fill the plane. Let the machine be $(Q, \Sigma, d, q_{\text{start}})$. The colours of the tile set are given by

$$C \;=\; \big(Q \times \{L, R\}\big) \;\cup\; \big(Q \times \Sigma\big) \;\cup\; \Sigma \;\cup\; \{\bullet, \square, \diamond, \not{\ell}\}.$$

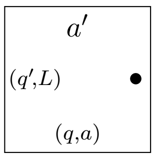
The set T consists of 5 types of tiles.

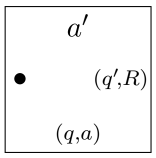
1. For each tape symbol, we have a tile that represents a cell of the tape on which the computation head is not located:

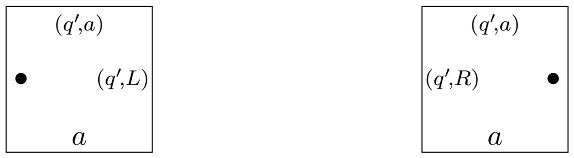
   Above and below them should be tiles that encode the same symbol; therefore the colours on the North and the South are chosen to be the same symbol of Σ.

2. For each $(q, a) \in \operatorname{Dom} d$, there is a tile that represents the computation head at the beginning of a computation step with control state q that reads a. Let $(a', m, q') = d(q, a)$. This tile also encodes the writing of a', a move in the direction $m \in \{L, R\}$ and a changes to the control state q'. Depending on the value of m, we either add the left $(m = L)$ or the right $(m = R)$ tile to the tile set T:

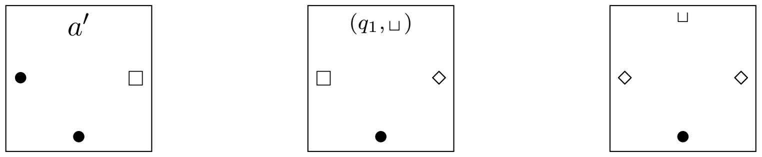
   

3. For each $(q, a) \in Q \times \Sigma$, there is a tile that represents the head at the end of the computation step. For left and right movements they are:



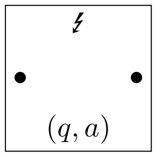
Note that only tiles of this and the above type have colours $Q \times \{L, R\} \subseteq C$, on the left or the right side. This implies that a tile of this type always has exactly one neighbour which is a tile of the previous type and vice versa. Also note that the colours $Q \times \Sigma \subseteq C$ only appear in tiles of this and the previous type. (There is only one exception, corresponding to a single tile that can only appear on the first row.) This implies that in each subsequent row, the position of the computation head will be in the desired place. It is not possible that a second computation head is created, because tiles of the previous type can only appear above tiles of this type.

4. The initial configuration is given by the state $q_{\text{start}} \in Q$ and a tape containing only blank symbols $\sqcup \in \Sigma$. By assumption, the machine does not halt immediately, and the head must move right. Let a' be the symbol that is written in this step and q_1 be the new state. To enforce a correct first row, the tile set contains



We choose t to be the leftmost tile.

5. For each $(q, a) \notin \operatorname{Dom} d$, (which can only appear in a terminating configuration), we add the tile



The colour ↯ does not appear in the South of any tile.

In a tiling of the plane each row has exactly one tile of the second and third type. Therefore, the control states encoded in these cells correspond to the to the control states of a valid computation history. Hence, if the computation never terminates, this tile set can fill the plane. Otherwise, the tile with the colour ↯ appears. No other tile fits above this tile, hence, in this case there is no (T, t)-tiling. □

4.6 Exercises

Exercise 15. The following problem is also called the *generalized Collatz* problem. Let (a, b, r_0) be a triple where $a = [a_1, \ldots, a_k]$ and $b = [b_1, \ldots, b_k]$ are lists of rational numbers of equal length and r_0 is a rational number. With each such triple we associate a computational process that operates on a single rational number r. Initially, $r = r_0$. In each step of the process, r is replaced by

$$r \cdot a_{\lfloor r \rfloor \bmod k} + b_{\lfloor r \rfloor \bmod k}.$$

The computation halts if $r = 1$. Show that the set of triples that halt is undecidable.

Appendix: The halting set for 2-tag systems is undecidable

Theorem. *The following set is undecidable*

$$\{\langle M, w\rangle : M \textit{ is a 2-tag system that halts on input } w\}.$$

Proof. We showed already that register machines can simulate Turing machines. We now show that 2-tag machines can simulate register machines. In this way, we show that there is a reduction from $\mathrm{H}_{\mathrm{TM},\varepsilon}$ to the set of the theorem.

It suffices to implement the three commands of exercise **??**. Let $a, b, \ldots, e$ be the variables of the register machine. For each line ℓ of the program, the tag system has variables $\mathtt{a}_\ell, \ldots, \mathtt{e}_\ell$ and $\mathtt{A}_\ell, \ldots, \mathtt{E}_\ell$ (for the lines with an `If` instruction, we need additional symbols). The alphabet also contains a fixed symbol $*$.

The tag-machine is started with the word

$$\mathtt{a}_\ell * \mathtt{A}_\ell * \mathtt{b}_\ell * \mathtt{B}_\ell * \ldots \mathtt{e}_\ell * \mathtt{E}_\ell *$$

In the simulation of a computation step of the register machine at line ℓ, the tag machine is initially in a state

$$\overbrace{\mathtt{a}_\ell ? \ldots \mathtt{a}_\ell ?}^{2^a \text{ times}} \mathtt{A}_\ell ? \overbrace{\mathtt{b}_\ell ? \ldots \mathtt{b}_\ell ?}^{2^b \text{ times}} \mathtt{B}_\ell ? \quad \ldots \quad \overbrace{\mathtt{e}_\ell ? \ldots \mathtt{e}_\ell ?}^{2^e \text{ times}} \mathtt{E}_\ell ?$$

At the places of the question marks can be any symbol; these symbols do not influence the computation because they are deleted before they can be scanned. Note that the start state corresponds to a state of the register machine where all variables are empty.

Assume line ℓ is an instruction $b := b + 1$, then for all $\mathtt{x} \in \{\mathtt{a}, \mathtt{A}, \ldots, \mathtt{e}, \mathtt{E}\} \setminus \{\mathtt{b}\}$, we choose

$$d(\mathtt{x}_\ell) = \mathtt{x}_{\ell+1} *$$

and for $\mathtt{b}_\ell$ we choose

$$d(\mathtt{b}_\ell) = \mathtt{b}_{\ell+1} * \mathtt{b}_{\ell+1} *$$

After replacing all symbols with index ℓ, the word of the tag machine is in the correct state to start the execution of line $\ell + 1$.

If a line ℓ contains `Stop`, then d is undefined for all symbols $\mathtt{x}_\ell$. It remains to explain how an `If`-instruction is implemented, and this is the hardest part. Consider a command

ℓ. `If` $b = 0$ `then goto line` m `else` $b := b - 1$ `and goto line` n

The rough idea. Recall that only the odd positions of the word are scanned. First we decrement b, in other words, we halve the number of symbol $\mathtt{b}_\ell$ by choosing $d(\mathtt{b}_\ell) = \dot{\mathtt{b}}_\ell$. If $b = 0$ this will make the length of the string odd, otherwise the length is even. While decrementing, we also make sure that every symbol is repeated twice, i.e., for all $\mathtt{x}_\ell \neq \mathtt{b}_\ell$ we choose $d(\mathtt{x}_\ell) = \dot{\mathtt{x}}_\ell \dot{\mathtt{x}}_\ell$. Then we go through the whole string and replace the symbols $\dot{\mathtt{x}}_\ell$ by $\mathtt{x}_n \mathtt{x}_m$. This strings has variables labeled by n in its odd positions, and variables labeled by m in its even positions. Now, the main idea: if the string has odd length, the last replacement of a symbol $\dot{\mathtt{x}}_\ell$ will delete the first symbol $\mathtt{x}_n$. The positions that were previously even, became odd, and they are all labeled by m. If $b > 0$, then this would not happen, and the odd positions still contain symbols labeled by n. Moreover, the string is such that the simulation of the next computation step of the register machine can start.

Now the detailed construction. To simulate a line ℓ that modifies a variable v, we divide the word in 4 parts which are schematically represented as

$$\mathtt{x}_\ell? \dots \mathtt{x}_\ell?\mathtt{v}_\ell? \dots \mathtt{v}_\ell?\mathtt{V}_\ell?\mathtt{y}_\ell? \dots \mathtt{y}_\ell?.$$

The leftmost part is the part that contains everything at the left of the symbols $\mathtt{v}_\ell$, the second part is the part that contains the repetitions of $\mathtt{v}_\ell?$, the third part, is the pair of symbols $\mathtt{V}_\ell?$, and the forth part is everything else.

In its first run through the string, we use copies of the symbols $\mathtt{z}$ given by $\dot{\mathtt{z}}$, $\mathtt{z}^>$ and $\mathtt{z}^=$ and the rules

$$\begin{aligned} d(\mathtt{x}_\ell) &= \dot{\mathtt{x}}_\ell* \qquad & d(\mathtt{V}_\ell) &= \mathtt{V}^=_\ell\mathtt{V}^>_\ell \\ d(\mathtt{v}_\ell) &= \dot{\mathtt{v}}_\ell \qquad & d(\mathtt{y}_\ell) &= \mathtt{y}^=_\ell\mathtt{y}^>_\ell. \end{aligned}$$

After scanning all odd positions we obtain the word

$$\dot{\mathtt{x}}_\ell* \dots \dot{\mathtt{x}}_\ell*\dot{\mathtt{v}}_\ell \dots \dot{\mathtt{v}}_\ell\mathtt{V}^>_\ell\mathtt{V}^=_\ell\mathtt{y}^>_\ell\mathtt{y}^=_\ell \dots \mathtt{y}^>_\ell\mathtt{y}^=_\ell.$$

Thus this halves the number of $\mathtt{v}$'s and makes the symbols in odd and even positions different in the two rightmost parts.

Let us now consider the case where $v = 0$. We obtain a string with a single $\dot{\mathtt{v}}_\ell$ symbol. In other words, the state above is

$$\dot{\mathtt{x}}_\ell* \dots \dot{\mathtt{x}}_\ell*\dot{\mathtt{v}}_\ell\mathtt{V}^>_\ell\mathtt{V}^=_\ell\mathtt{y}^>_\ell\mathtt{y}^=_\ell \dots \mathtt{y}^>_\ell\mathtt{y}^=_\ell$$

This string has odd length. The second time the tag machine goes through the string, it uses

$$\begin{aligned} d(\dot{\mathtt{x}}_\ell) &= \mathtt{x}_n\mathtt{x}_m \qquad & d(\mathtt{V}^=_\ell) &= *\mathtt{V}_m* \\ d(\dot{\mathtt{v}}_\ell) &= \mathtt{v}_n\mathtt{v}_m \qquad & d(\mathtt{y}^=_\ell) &= \mathtt{y}_m*. \end{aligned}$$

The replacement of the last symbol $\mathtt{y}^=_\ell$ deletes the first symbol $\mathtt{x}_n$. We obtain the string

$$\cancel{\mathtt{x}_n}\mathtt{x}_m \dots \mathtt{x}_n\mathtt{x}_m\mathtt{v}_n\mathtt{v}_m*\mathtt{V}_m*\mathtt{y}_m* \dots \mathtt{y}_m*$$

The string is now in the form

$$\mathtt{x}_m? \dots \mathtt{x}_m?\mathtt{v}_m?\mathtt{V}_m?\mathtt{y}_m? \dots \mathtt{y}_m?$$

and the next step of the simulation can be started. Note that this string represents a register machine where $v = 0$.

The case $v > 0$ goes in a similar way, and it is even simpler, because the first symbol $\mathtt{x}_n$ is not removed. We choose $d(\mathtt{V}^>_\ell) = \mathtt{V}_n*$ and $d(\mathtt{y}^>_\ell) = \mathtt{y}_n*$. Note that the represented value of v has decreased by one. □

Solutions

Exercise 1. Assume $A_i = \{a_{i,1}, a_{i,2}, \dots\}$ for all i. Again we use the list for pairs of natural numbers:

$$\bigcup_{i \in \mathbb{N}} A_i = \{a_{1,1}, a_{1,2}, a_{2,1}, a_{3,1}, a_{2,2}, a_{1,3}, \dots\}\,.$$

By definition $A^* = \{\varepsilon\} \cup A \cup A^2 \cup \dots$ and each set A^k is countable.

Exercise 2. Every such sequence is given by appending infinitely many 0's after some bitstring x. Because the set of bitstrings is countable, also this set is countable.

Exercise 3. Consider the table whose rows are values of $\varphi(n, i)$ for $i = 1, 2, \dots$, for example:

n \ i	1	2	3	4	5	...
1	**3**	4	2	1	1	...
2	0	**0**	0	1	2	...
3	0	5	**5**	5	5	...
4	0	1	0	**1**	0	...
5	8	7	6	5	**4**	...
⋮			⋮			
diagonal + 1	**4**	**1**	**6**	**2**	**5**	...

The diagonal plus 1 defines a function that does not appear in the table.

Exercise 4. We run the checker on all strings in lexicographic order. We define $\varphi(n, \cdot)$ to be the function that is defined by the nth program that was accepted by the checker. Thus, for all n, the function $\varphi(n, i)$ is defined on all i. From 3 we obtain a total computable function that differs from $\varphi(m, \cdot)$ for all m. By construction, this function can not be implemented in the language.

Exercise 5. Let $g(x) = 1 - M(\langle M \rangle)$ if $x = \langle M \rangle$ for some Turing machine M and if $M(\langle M \rangle)$ is defined, and let $g(x)$ be undefined otherwise. Now assume that h is a computable function. (Recall that computable functions are total.) Then $h(x) \neq g(x)$ if x equals $\langle M \rangle$ for the machine that computes h. Hence, if h can not be an extension of g.

Exercise 6. If A is finite, then it is easy to define such a function. Assume that A is infinite. Hence, we can enumerate the set as a stream and this stream is infinite. Let $f(i)$ be the ith element of the stream. Note that f is computable.

Exercise 7. If A is empty, then this is trivially true. Otherwise, by exercise 6 there exists a computable function f such that A is the image of f. Let P be the predicate such that $P(n, x)$ is true if $x = f(n)$ and false otherwise. We have that $\exists n \colon P(n, x)$ if and only if x is in the image of f.

Exercise 8. 4. Let A be a recognizable language that is not computable, for example, the halting problem $\mathrm{H_{TM}}$. Let ${}^{\mathsf{c}}A$ be its complement. The set that satisfies the conditions is given by $X = A \oplus^{\mathsf{c}} A$. Assume this set was recognizable. By item 2, also ${}^{\mathsf{c}}A$ is recognizable, and hence A is decidable by item 3, contradicting Theorem 4. Thus, X is not recognizable. For the same reason, ${}^{\mathsf{c}}X =^{\mathsf{c}} A \oplus A$ is also not recognizable.

An alternative solution, is to use item 5 and show that a random set satisfies this property with probability 1.

Exercise 10. By assumption on C, there exist $y_{\text{in}} \in C$ and $y_{\text{out}} \notin C$. The algorithm for the reduction from B to C works as follows: on input x, it first runs the decider for B, and if $x \in B$, it outputs y_{in}, otherwise, it outputs y_{out}.

Exercise 11. Let $B = \mathrm{H}_{\mathrm{TM}}$. B is enumerable and undecidable. If ${}^{\mathsf{c}}B \leq_m B$, then ${}^{\mathsf{c}}B$ is also enumerable, and by exercise 8.3, B is decidable. A contradiction.

Exercise 12. If B and C reduce to X using the same reduction f, then $B = C$, because

$$x \in B \iff f(x) \in X \iff x \in C.$$

Because there are countably many computable functions f, at most countably many subsets of Σ^* reduce to X. Because Σ^* is infinite, this set has uncountably many subsets, hence, there must be some set that does not reduce to X.