University of Warsaw
Faculty of Philosophy and Sociology
Institute of Philosophy

Michał Wrocławski

Representations of Numbers and their Computational Properties

PhD Thesis

Supervisor: Prof. Stanisław Krajewski

Assistant Supervisor: Dr Dariusz Kalociński

Warsaw, 2019

Abstract

The distinction between computable and uncomputable functions on natural numbers occupies a very important place in the theory of computability. A function is considered computable if there exists an algorithm which can read any valid arguments as input and return the correct value of the function on these arguments as output. Little consideration is usually given to the way in which the input and output numbers are represented.

In this thesis we follow the approach presented by Stewart Shapiro in his article 'Acceptable notation'. We assume that computations are performed directly on numerals, and only indirectly on numbers represented by these numerals. Depending on the choice of a representation, different functions can become computable or uncomputable. We examine properties of various representations and try to establish correlations between these properties. We consider in what cases computability of some functions implies computability of others. We show, among others, that computability of successor, multiplication, or exponentiation does not imply computability of the rest of these functions.

Our notion of representation differs from that used by Shapiro, since we also allow the possibility that certain numbers can be represented by more than one numeral. We attempt to answer the question how such extension of the concept of representation affects its properties. We manage to prove certain theorems which are generalised versions of Shapiro's results. In particular, we show that functions computable in every representation are exactly constant functions and projections.

We show that representations with computable identity (including unambiguous representations) manifest much more regular properties than the other ones. We prove certain theorems which are valid for such representations, but do not hold in general case (we provide counterexamples to them).

We also show that not all notions translate well to such numeral-based approach to computability. In particular, the notion of Turing reducibility exhibits some unexpected properties in this context. We consider various possible ways of defining this notion within our framework and show their defficiencies.

We also analyse philosophical implications of the perspective on computability based on the notion of representation. We discuss how such perspective affects our interpretation of Church's thesis.

Streszczenie

Rozróżnienie na obliczalne i nieobliczalne funkcje na liczbach naturalnych zajmuje bardzo ważne miejsce w teorii obliczeń. Funkcja jest uznawana za obliczalną, jeżeli istnieje algorytm, który może wczytać dowolne poprawne argumenty na wejściu i zwrócić właściwą wartość funkcji dla tych argumentów na wyjściu. Niewiele uwagi przywiązuje się zwykle do tego, w jaki sposób reprezentowane są liczby na wejściu i wyjściu.

W tej rozprawie podążamy za podejściem zaprezentowanym przez Stewarta Shapiro w jego artykule „Acceptable notation". Zakładamy, że obliczenia wykonywane są bezpośrednio na liczebnikach, zaś jedynie pośrednio na liczbach przez te liczebniki reprezentowanych. W zależności od wyboru reprezentacji, różne funkcje mogą się stać obliczalne albo nieobliczalne. Badamy własności różnych reprezentacji i próbujemy ustalić zależności między tymi własnościami. Rozważamy, w jaki sposób obliczalność jednych funkcji pociąga za sobą obliczalność innych. Pokazujemy m.in., że obliczalność następnika, mnożenia albo potęgowania, nie implikuje obliczalności pozostałych spośród tych funkcji.

Nasze pojęcie reprezentacji różni się od tego używanego przez Shapiro, gdyż dopuszczamy również możliwość, by pewne liczby były reprezentowane przez więcej niż jeden liczebnik. Staramy się odpowiedzieć na pytanie, w jaki sposób takie rozszerzenie pojęcia reprezentacji wpływa na jego własności. Udaje nam się udowodnić pewne twierdzenia, które są uogólnionymi wersjami rezultatów Shapiro. W szczególności pokazujemy, że funkcje obliczalne w każdej reprezentacji to dokładnie funkcje stałe oraz rzutowania.

Pokazujemy, że reprezentacje z obliczalną identycznością (w tym reprezentacje jednoznaczne) przejawiają dużo bardziej regularne własności niż pozostałe. Dowodzimy pewnych twierdzeń, które są prawdziwe dla takich reprezentacji, ale w ogólnym przypadku nie zachodzą (podajemy dla nich kontrprzykłady).

Pokazujemy również, że nie wszystkie pojęcia dają się dobrze przełożyć na takie oparte na liczebnikach podejście do obliczalności. W szczególności, pojęcie redukowalności w sensie Turinga przejawia w tym kontekście pewne nieoczekiwane własności. Rozważamy różne możliwe sposoby zdefiniowania tego pojęcia w zgodzie z naszym podejściem i wskazujemy na ich niedostatki.

Analizujemy również filozoficzne implikacje takiego podejścia do obliczalności opartego na pojęciu reprezentacji. Omawiamy kwestię, w jaki sposób taka perspektywa wpływa na naszą interpretację tezy Churcha.

Acknowledgments

I would like to thank my supervisor prof. Stanisław Krajewski and my assistant supervisor dr Dariusz Kalociński for all their help and support in preparing this thesis. They have provided a lot of constructive comments and suggestions which have helped me immensely to improve this thesis. I am deeply grateful to them.

I am also very thankful to prof. Marcin Mostowski who is unfortunately no longer among us. He was the one who encouraged me to pursue the path of logic and who inspired me to carry out research in representations of numbers.

Contents

1 Introduction

1.1 Historical background

In this section we provide a historical background for issues discussed in this thesis. We discuss some early attempts at formulating general frameworks for solving problems and show how in 20th century it became clear to mathematicians and logicians that some problems cannot be solved in a mechanical way.[1]

1.1.1 Leibniz

It might appear to be a very tempting perspective to invent a method, preferably a purely mechanical one, of solving all possible problems. It is well-known that such a postulate was raised by Gottfried Leibniz (1646–1716)[2] who proposed establishing both a universal language (*characteristica universalis*) and a universal logical problem-solving framework (*calculus ra-*

[1] In addition to sources cited below, this subject is also thoroughly discussed in [36].

[2] The reconstruction of ideas of Leibniz provided below is based on [35, pp. 101–120], [3] and [51].

tiocinator). In his often quoted words, Leibniz claimed that:[3]

> if controversies were to arise, there would be no more need of disputation between two philosophers than between two accountants. For it would suffice to take their pencils in their hands, to sit down with their slates and to say to each other (with a friend as witness, if they liked): Let us calculate.

Leibniz believed that such a universal calculus would allow us to reason in metaphysics and morals just as we do in mathematics.[4] While contemporary symbolic logic had obviously not yet been developed in Leibniz's time, he mentioned Euclid's axioms for geometry and syllogisms in logic in his writings about his desired system for deduction.

While the 17th century has witnessed significant new developments in mathematics and natural sciences, logic was still being neglected then. It was considered obsolete and useless, a relic of Aristotle's philosophy. Leibniz was one of few thinkers of those times who understood its importance. He valued the infallible nature inherent to the art of logic.

The assessment of Leibniz's impact on logic is ambiguous. Couturat [13] acknowledged ingenuity of Leibniz's ideas in logic and his achievements, and yet at the same time criticised him for being too attached to the old ways of Aristotle to be able to fully pursue new directions. On the other hand, Lenzen argues (in [33], [34]) that Leibniz developed what could be considered a full Boolean algebra of sets. According to Parkinson, Leibniz was the first symbolic logician, though unfortunately his writings in this areas were only published in the 19th century after similar ideas had already been developed by others.

The situation is even more complicated because Leibniz's writings on logic are often incomplete. In various places, Leibniz pursued similar ideas, yet details of their development changed over time. As a result, it could be difficult to combine them into one consistent well-developed theory.

Leibniz believed that axioms were supposed to be propositions regarding identity of objects and that they could be obtained by analysing concepts

[3]Translation by Bertrand Russell in [51, p. 200], the quote comes from Leibniz's work *De arte characteristica ad perficiendas scientias ratione nitentes* from 1684, which can be found in [32].

[4]This is not a very unusual idea. Spinoza made a similar attempt with his *Ethics: Demonstrated in Geometric Order* [59].

until they are simple enough. Further propositions could be obtained from axioms by using rules of inference. In addition to formulating the very idea of characteristica universalis, Leibniz also contributed some results to it. He defined concepts such as identity, containment (a subset), converse containment (a superset), conjunction (intersection of sets), negation (complement of a set) and possibility (not being equal to the empty set). He also formulated some laws which are presented below in contemporary notation. Capital letters A, B, C are interpreted as representing concepts. The function φ assigns to each concept all possible individuals that fall under this concept and $\overline{\varphi(A)}$ means the complement of $\varphi(A)$.

1. $\varphi(A) \subseteq \varphi(A)$,
2. $\varphi(A) \subseteq \varphi(B) \wedge \varphi(B) \subseteq \varphi(C) \Rightarrow \varphi(A) \subseteq \varphi(C)$,
3. $\varphi(A) \subseteq \varphi(B \cap C) \Leftrightarrow \varphi(A) \subseteq \varphi(B) \wedge \varphi(A) \subseteq \varphi(C)$,
4. $\overline{\overline{\varphi(A)}} = \varphi(A)$,
5. $\varphi(A) \cap \overline{\varphi(B)} = \emptyset \Leftrightarrow \varphi(A) \subseteq \varphi(B)$,
6. $\varphi(A) \subseteq \varphi(B) \wedge \varphi(A) \neq \emptyset \Rightarrow \varphi(B) \neq \emptyset$.

In the 19th and 20th century, it became much more common to think about logic the way Leibniz did.

1.1.2 Boole

The ideas of Leibniz were further pursued by George Boole (1815–1864) who developed them in his work *Mathematical Analysis of Logic* [6] and later in *The Laws of Thought* [7].[5] Boole claimed that logic should be closely related to mathematics, not philosophy, pointing at similarities between logic and algebra.

In the introduction to *Mathematical Analysis of Logic*, Boole emphasised that in algebra

> the validity of the processes of analysis does not depend upon the interpretation of the symbols which are employed, but solely upon the laws of their combination.

[5] Our reconstruction of Boole's ideas is based mainly on [35, pp. 143–164.].

The nature of objects in question is irrelevant. Thus, the same process of reasoning can, depending on its interpretation, lead to solutions of several problems of entirely different areas. Boole strove for a similar treatment of logic and he managed to construct such a theory which is now referred to as Boolean algebra.[6]

In *The Laws of Thought* Boole introduced the idea of what we would now call a formal theory. It consisted of:

1. terms for classes $x, y, z, ...$,
2. the empty class 0,
3. the universe class 1,
4. a relation symbol of identity $=$,
5. symbols for operations: addition $+$, multiplication $\cdot$ and subtraction $-$, interpreted respectively as union, intersection and complement of classes.

It did not have any quantifiers, though.

Boole also formulated the following laws governing the use of these operations:

1. $x + y = y + x$.
2. $x \cdot y = y \cdot x$,
3. $x^2 = x$,
4. $z \cdot (x + y) = z \cdot x + z \cdot y$,
5. $x - y = -y + x$,
6. $z \cdot (x - y) = z \cdot x - z \cdot y$,

supplemented by two rules of inference:

1. adding or subtracting equals from equals gives equals,
2. multiplying equals by equals gives equals.

Boole's ideas were another important step in developing contemporary algebra and logic.

[6]However, the name 'Boolean algebra' was probably coined in 1913 by Henry Maurice Scheffer, according to [25].

1.1.3 Hilbert

Symbolic logic developed in the 19th and 20th century contributed a lot to bringing Leibniz's idea to life but its scope of success was very limited.

In 1910, David Hilbert formulated a series of problems (Hilbert's problems) to be solved by mathematicians. The tenth of these problems concerned solvability of Diophantine equations, i.e polynomial equations with a finite number of unknowns and all integer coefficients. The problem consisted in providing an algorithmic procedure to determine, for each such equation, whether it has integer solutions or not.[7]

In 1928, David Hilbert, together with Wilhelm Ackermann, formulated another problem of a similar kind, *Entscheidungsproblem* (in German it means: 'decision problem').[8] He asked to provide an algorithm which takes a statement of first-order logic and a finite number of axioms (in addition to the usual logical axioms) as input and determines whether the given sentence is valid in every structure satisfying these axioms.

Initially, it seemed obvious to Hilbert and other mathematicians that such algorithm had to exist, the only difficulty lied in finding it. It was only under the influence of Gödel's incompleteness theorems that in the 1930s they began to search for proof that it cannot exist.

Finally, the impossibility of such algorithm has been proved. The negative answer to Hilbert's question was given independently by Alonzo Church [10] and Alan Turing [62]. To demonstrate this result, Church made use of λ-calculus, while Turing of 'automatic machines', currently known as Turing machines.[9]

[7] This problem was not solved until 1970, when Yuri Matiyasevich showed that no such procedure exists (see: [37]).

[8] This subsection is largely based on [35, pp. 243–268, 293–314].

[9] Both these models of computation, and also several others, have been proved to be equivalent. The supposition that the intuitive notion of computability and the notion expressed by each of these models are equivalent, is known as Church's thesis or the Church-Turing thesis. Both Church's and Turing's negative answers to Hilbert's question rest upon the assumption that this thesis is correct, which is commonly believed to be true.

1.2 Solvable and unsolvable problems

It has been stated in the previous section that not every problem can be solved by an algorithm. A problem is called solvable if there is an algorithm which halts for every admissible input and returns the correct answer, otherwise a problem is unsolvable. Problems are often represented by sets. Then the problem consists in determining, for any given object (or finite sequence of objects), whether this object (sequence of objects) belongs to the given set.

Let us consider Hilbert's *Entscheidungsproblem*. It can be represented as the set of all first-order logic sentences valid in every structure satisfying the given axioms. If there existed an algorithm which could solve it, then it would be able to decide whether any first-order formula (taken as the input) belongs to this set or not. Then, it would return a Boolean value $TRUE$ or $FALSE$ as the output.

Hilbert's tenth problem and *Entscheidungsproblem* are not the only unsolvable problems in mathematics. Here are more examples:

1. Halting problem: let $(M_i)_{i\in\mathbb{N}}$ be a recursive enumeration of all Turing machines (could also be computer programs in a specified programming language). Let:
$$K = \{(i, x) : M_i(x) \downarrow\},$$
where $M_i(x) \downarrow$ means that the machine M_i halts if the input is x. For any given pair of natural numbers (i, x) decide whether this pair belongs to K or not.

2. For a given consistent recursively enumerable axiomatic theory T containing elementary arithmetic and for any arithmetical sentence φ decide whether $T \vdash \varphi$. It is known that such algorithm does not exist thanks to Gödel's first incompleteness theorem [21].

3. Post correspondence problem [44]: let Σ be an alphabet with at least two symbols. Two finite lists of words over A are given: $\alpha_1, ..., \alpha_n$ and $\beta_1, ..., \beta_n$. The problem consists in deciding whether there exists a sequence of indices $i_1, ..., i_k$ such that $k \geq 1$, $1 \leq i_j \leq n$, for all j, and
$$\alpha_{i_1} \frown ... \frown \alpha_{i_k} = \beta_{i_1} \frown ... \frown \beta_{i_k}.$$

4. Finite satisfiability [61]: determine for any first-order sentence whether it is satisfiable in a certain finite model. The undecidability of this problem is stated by Trakhtenbrot's theorem.

While a problem can be labelled solvable, when we speak of sets or functions, it is probably more common to call them computable or decidable.

It is a well-known and easy conclusion from the theory of cardinal numbers that nearly all sets and functions are not computable. This is because all algorithms are finite inscriptions consisting of symbols from a finite alphabet — there can only be countably many such inscriptions, thus there can only be countably many computable sets and functions. There are, however, uncountably many sets and functions in general — therefore uncountably many of them are uncomputable. Of course, not all uncomputable functions or sets are of any interest to us, but some are, including those enumerated earlier in this section.

1.3 Relativisation of the notion of computability

All computations deal with their respective objects via some kind of a representation. The basic type of objects that we are going to deal with here are natural numbers. When we speak of computability of functions on natural numbers, we usually understand them as functions on numbers represented by decimal numerals. In some cases we might prefer to use binary or perhaps even Roman numerals, but that is of little concern to us since we can easily translate between each of these representations. If we can compute the value of a certain functions on decimal numerals, then we can also do it on binary numerals or Roman numerals, and *vice versa*.

Some representations can, however, be very different from those commonly used. Any assignment of numbers to numerals could serve as a representation, and every such representation leads to a certain notion of computability. This notion does not have to be in any way equivalent to that defined by Turing since representations differ as to what functions and sets can be computed in them. We shall provide a more formal definition of a representation which shall be slightly more restrictive than what we have sketched here but it will be based on the same intuition.

Let us consider Hilbert's *Entscheidungsproblem*. Let $(\varphi_i)_{i\in\mathbb{N}}$ be a recursive enumeration of all formulas of first-order logic. The set of indices of logically provable formulas is not computable. Let $i_0, i_1, ...$ be the indices of all such formulas and $j_0, j_1, ...$ — the indices of all unprovable formulas. Now suppose that even numerals (i.e. even according to the standard interpretation) represent respectively numbers $i_0, i_1, ...$ and odd numerals — numbers $j_0, j_1, ...$ Then the usual sequence of numerals (not numbers) $0, 1, 2, 3, ...$ would represent the sequence of numbers $i_0, j_0, i_1, j_1, ...$

However, in such a representation the *Entscheidungsproblem* would be solvable (at least for the case where the set of assumptions is empty, but we could easily construct similar representations for cases where it is not). For any natural number, in order to determine whether it is an index of a provable first-order formula, one would simply need to check whether it is represented by an even numeral.

We have thus concluded that the *Entscheidungsproblem* is unsolvable in the standard decimal representation and yet it is solvable in the non-standard representation presented above. In this thesis we are going to consider various representations of natural number and their computational properties. These are some of the problems we are going to tackle:

1. Introduce a formal notion of a representation and of computability of a function, set or relation in a given representation (Chapter 2).

2. Define criteria according to which some representations can be considered similar to each other, or certain representations can be considered better than other ones. Establish correlations between these criteria (Chapters 3 and 6).

3. Investigate which functions (if any) are computable in no, some, or every representation (Chapter 4).

4. Establish correlations between computability of basic functions and relations, including successor, addition, multiplication and exponentiation, in various representations of natural numbers (Chapter 4).

5. Consider the notion of Turing reducibility relativised to a given representation (Chapter 7).

6. Investigate if there are any particularly strong or crucial assumptions regarding representations which give us significant insight into proper-

ties of those representations which satisfy these assumptions (throughout the thesis, but especially in Chapter 5).

7. Consider philosophical implications of the perspective on computability which allows existence of various representations of numbers, especially regarding Church's thesis (Chapter 8).

Considerations contained in this thesis have been inspired by Stewart Shapiro's article 'Acceptable notation' [52]. We have decided to modify his terminology, though. In particular, we have proved generalised or modified versions of some of Shapiro's results, in particular Theorems 4.3, 4.5, 4.11, 4.14, 4.23, 5.11. Part of terminology used in this thesis comes from Konrad Zdanowski's unpublished conference slides [65], but in a modified version to better suit our needs.

Some of our results contained in this thesis come from papers [63] and [64] (accepted for publication). For the sake of completeness of our presentation of the subject, we repeat their proofs here, possibly with some modifications. Each of these theorems is accompanied by a reference to the original paper.

Throughout this thesis we shall adopt the following convention: if $f : A \to B$ and $A' \subseteq A$, then $f(A') = \{f(n) : n \in A'\}$.

2

Formalisation of the concept of representation

In this chapter we want to establish formal definitions of some key concepts regarding representations. We shall begin with defining a representation, then proceed to define the notion of computability relative to a given representation. We also wish to prove some basic properties of these notions.

The notions introduced in this chapter shall primarily be applied to natural numbers. However, they can easily be utilised and examined with regard to other types of objects, including other countable sets of numbers (such as rational numbers or algebraic numbers), ordinal numbers,[1] countable families of sets, various algebras etc.

2.1 Definition and examples of representations

Let us define the notion of a representation of a set of objects.

Definition 2.1 *Let Z be a countable infinite set and Σ — a finite alphabet.*

[1] Notations for ordinal numbers were considered by Church [11] and Kleene [28], see also [50].

We shall call (S, σ) a representation of Z if $S \subseteq \Sigma^*$ is a computable set and $\sigma : S \to Z$ is a surjection.

The elements of Σ shall be called digits or symbols and the elements of S — numerals, inscriptions or strings.

Also, this definition could easily be extended to representations of finite sets. However, we shall not consider such representations here.

For every $\alpha \in S$, if $\sigma(\alpha) = n$, then we say that α represents n, and that n is the interpretation of α. If $S' \subseteq S$, and $\sigma(\alpha) = n$, for every $\alpha \in S'$, then we can also say that n is represented by the set S'.

If $\alpha, \beta \in S$ and $\sigma(\alpha) = \sigma(\beta)$, then we can write $\alpha =^\sigma \beta$. This differs from writing $\alpha = \beta$ which means that they are the same numerals.

In the above definition of a representation, Z needs to be a countable set because we are unable to represent uncountably many objects with finite sequences over a finite (or even countable) alphabet.[2]

We would like to give some examples of representations:

Definition 2.2 *The unary representation of $\mathbb{N}$ is a pair (S, σ) such that $\Sigma = \{\overline{1}\}$, S is the set of all finite sequences comprised of $\overline{1}$ and the empty word ε, and the function σ is defined as follows:*

$$\sigma(\varepsilon) = 0,$$

$$\textit{if } \sigma(\alpha) = n, \textit{ then } \sigma(\alpha \frown \overline{1}) = n + 1.$$

Definition 2.3 *The standard (decimal) representation of $\mathbb{N}$ is a pair (N, σ_{st}) such that $\Sigma = \{\overline{0}, \overline{1}, ..., \overline{9}\}$, N is the set of all standard decimal numerals (i.e. the set consisting of the numeral $\overline{0}$ and of all finite sequences of digits from Σ which do not begin with $\overline{0}$), and the function σ is defined as follows:*

$$\sigma(\overline{a_n}...\overline{a_0}) = \sum_{i=0}^{n} a_i \cdot 10^i,$$

We shall adopt a convention that N always refers to the set of standard decimal numerals. Furthermore, unless explicitly stated otherwise, we always

[2]However, it is possible to 'imitate' an infinite set of symbols $\{a_i : i \in \mathbb{N}\}$ using only a finite alphabet. It suffices to take $\Sigma = \{a, \overline{1}\}$ and imitate every symbol a_i with $a \underbrace{\overline{1}...\overline{1}}_{i \text{ times}}$.

assume that for any natural number n, by $\overline{n}$ we refer to the standard numeral representing this number, i.e. for any natural number n:

$$\sigma_{st}(\overline{n}) = n.$$

This convention shall also extend to terms containing arithmetical symbols in such a way that e.g. the term $\overline{7 + 5 \cdot 2}$ shall refer to numeral $\overline{17}$, which represents number 17 in the standard representation.

Definition 2.4 *Let (S, σ) be a representation of Z. We shall say that this representation is unambiguous if for every $n \in Z$ there exists exactly one numeral $\alpha \in S$ such that $\sigma(\alpha) = n$. Otherwise we shall call the representation ambiguous.*

The unary and standard decimal representations of natural numbers defined as above are unambiguous.

Corollary 2.5 *For every representation (S, σ) of a set Z we are able to:*

1. *for each sequence of symbols over the given alphabet, decide whether it is a numeral of the considered representation,*
2. *generate an enumeration (without repetitions) of all numerals of this representation (which of course does not have to be identical with the standard order of numbers): ζ_0, ζ_1, ...*

2.2 Computability of functions in representations

In this section we shall consider what it means that a certain function, set or relation is computable in a given representation.

Definition 2.6 *Let (S, σ) be a representation of Z and let $f : Z^n \to Z$. Then f^σ shall denote the class of all functions $F : S^n \to S$ such that for any $\alpha_1, ..., \alpha_n, \beta \in S$ the following condition is satisfied:*

$$F(\alpha_1, ..., \alpha_n) = \beta \Rightarrow f(\sigma(\alpha_1), ..., \sigma(\alpha_n)) = \sigma(\beta).$$

Each function from the class f^σ shall be called a representation of f in (S, σ). If any of these functions is computable, then we shall say that f^σ is computable or that f is computable in (S, σ).

Obviously, if a representation is ambiguous, then there are multiple functions in f^σ. It is possible that some of them are computable, and some are not.

There is going to be a certain ambiguity when we talk of computing f^σ. Unless explicitly stated otherwise, it shall be synonymous with computing any function from the class f^σ. However, sometimes, when a concrete function from this class has already been specified, it can refer to computing this specific function.

Perhaps it could be helpful to think about it in a similar way as about indefinite integrals. While the indefinite integral of a function f is defined as the class of all functions whose derivative is f, this expression can also be used to refer to any specific function from that class.

Definition 2.7 *For any $R \subseteq Z^n$, the characteristic function of the relation R is the function χ_R such that for any $a_1, ..., a_n \in Z$ the following holds:*

$$\chi_R(a_1, ..., a_n) = TRUE \Leftrightarrow (a_1, ...a_n) \in R,$$

$$\chi_R(a_1, ..., a_n) = FALSE \Leftrightarrow (a_1, ...a_n) \notin R.$$

In particular:

$$\chi_=(a_1, a_2) = TRUE \Leftrightarrow a_1 = a_2,$$
$$\chi_=(a_1, a_2) = FALSE \Leftrightarrow a_1 \neq a_2.$$

In the above definition, $TRUE$ and $FALSE$ are not elements of Z (see: Theorem 2.9 and the discussion preceding it).

Note also that, in particular, this definition can be applied to characteristic functions of functions interpreted as relations in the usual way, i.e if $f : Z^n \to Z$, then:

$$\chi_f = \{(a_1, ..., a_n, b) : f(a_1, ..., a_n) = b\}.$$

When χ_A or χ_R is computable, we can also say that the set A or relation R is computable, without explicitly referencing the characteristic function.[3]

Throughout this paper, whenever we speak of functions, unless explicitly stated otherwise, we refer only to functions whose both arguments and values are numbers (in particular they are not logical values).

[3]This convention might be confusing when talking about characteristic functions of functions, though, due to Theorem 2.10.

Definition 2.8 *Let (S, σ) be a representation of Z and let $R \subseteq Z^n$. Then $R^\sigma \subseteq S^n$ shall be defined in the following way:*

$$(\alpha_1, ..., \alpha_n) \in R^\sigma \Leftrightarrow (\sigma(\alpha_1), ..., \sigma(\alpha_n)) \in R,$$

for all $\alpha_1, ..., \alpha_n \in S$. The set R^σ shall be called the representation of R in (S, σ). We say that R is computable in (S, σ) if R^σ is computable.

It is important to emphasise that $TRUE$ and $FALSE$ are not symbols from the alphabet Σ, nor from the set S, but they are additional symbols representing logical values. While it is not uncommon to identify logical values with some objects from the universe, such as numbers 1 and 0, in this case such approach would be problematic. The reason for this is that each number can be represented by many numerals, possibly by an infinite uncomputable subset of all numerals. We, on the other hand, want to define such notion of computability, according to which if a certain relation is computable, we can know for any sequence of numerals, whether they represent a sequence of numbers belonging to the set in question. The following theorem shows that the notion of computability of a relation (in a representation) is not equivalent to computability of a numerical function defined in an analogous way (for example, by identifying $TRUE$ with 1 and $FALSE$ with 0).

Theorem 2.9 *There exist (S, σ) — a representation of $\mathbb{N}$ and $A \subseteq \mathbb{N}$ such that χ_A is not computable in (S, σ), but the following function is:*

$$f_A(n) = \begin{cases} 1 & \text{if } n \in A, \\ 0 & \text{if } n \notin A. \end{cases}$$

Proof.

Let $B \subseteq \mathbb{N}$ be uncomputable in the standard representation, S — the standard set of numerals and let:

$$\sigma(\overline{2n}) = \begin{cases} 1 & \text{if } n \in B, \\ 0 & \text{if } n \notin B. \end{cases}$$

Assign all the other numbers to odd standard numerals in any surjective way. Let $A = \{1\}$. Then for any $n \in \mathbb{N}$ the following holds:

$$\overline{2n} \in A^\sigma \Leftrightarrow \sigma(\overline{2n}) = 1 \Leftrightarrow n \in B.$$

Since B is not computable in the standard representation, it follows that χ_A cannot be computable in (S, σ). However, f_A is computable in (S, σ) because we can define f_A^σ in the following way:

$$f_A^\sigma(\overline{2n}) = \overline{2n}$$

and

$$f_A^\sigma(\overline{2n+1}) = \overline{k},$$

where $\overline{k}$ is any numeral representing 0.

■

The following theorem establishes relation between two types of computability defined in this chapter:

Theorem 2.10 ([63]) *Let (S, σ) be a representation of Z and $f : Z^n \to Z$, for a certain $k \in \mathbb{N}$. If χ_f is computable in (S, σ), then f is computable in it as well. If, additionally, $\chi_=$ is computable in this representation, then the implication in the opposite direction also holds.*

Proof. Let (S, σ) be a representation of Z and f be a function. Let us assume that χ_f is computable in this representation. We shall prove that f is also computable. Suppose we want to compute the value of $f^\sigma(\alpha_1, ..., \alpha_n)$ for some $\alpha_1, ..., \alpha_n \in S$. We take a computable enumeration of all numerals $(\zeta_i)_{i \in \mathbb{N}}$ and we check them one by one until we find such ζ_i that

$$\chi_f^\sigma(\alpha_1, ..., \alpha_n, \zeta_i) = TRUE.$$

Then

$$\zeta_i = f^\sigma(\alpha_1, ..., \alpha_n)$$

is the value of the function we were looking for.

Now suppose that $\chi_=$ and f are computable in (S, σ). We shall prove that χ_f is computable as well. We want to find the value of $\chi_f(\alpha_1, ..., \alpha_n, \beta)$ for certain $\alpha_1, ..., \alpha_n, \beta \in S$. Since f is computable, we can find a numeral ζ_i such that

$$f(\alpha_1, ..., \alpha_n) =^\sigma \zeta_i.$$

Then:

$$\chi(\alpha_1, ..., \alpha_n, \beta) = TRUE \Leftrightarrow \beta =^\sigma \zeta_i,$$
$$\chi(\alpha_1, ..., \alpha_n, \beta) = FALSE \Leftrightarrow \beta \neq^\sigma \zeta_i.$$

■

3

Criteria for similarity of representations

It is possible that various representations of a certain set can have similar computational properties. In this chapter we shall define criteria for determining whether two representations are similar from a certain perspective or not.

3.1 Isomorphism and strong isomorphism

The first criteria to consider are isomorphism and strong isomorphism. They are related to the idea of being able to translate between representations.

Definition 3.1 *Representations (S, σ) and (T, τ) of Z are isomorphic if there exist computable functions (translations) F_τ^σ, F_σ^τ such that:*

$$\forall_{\alpha \in S} \exists_{\beta \in T} \ (F_\tau^\sigma(\alpha) = \beta \wedge \sigma(\alpha) = \tau(\beta)),$$

$$\forall_{\beta \in T} \exists_{\alpha \in S} \ (F_\sigma^\tau(\beta) = \alpha \wedge \sigma(\alpha) = \tau(\beta)).$$

If (S, σ) and (T, τ) are isomorphic, then we shall sometimes write:

$$(S, \sigma) \cong (T, \tau).$$

This criterion can be modified in the following way:

Definition 3.2 *Representations (S, σ) and (T, τ) are strongly isomorphic if there exists a computable function*

$$Id_\tau^\sigma : S \times T \to \{TRUE, FALSE\}$$

such that for any $a \in S, b \in T$:

$$Id_\tau^\sigma(\alpha, \beta) = TRUE \Leftrightarrow \sigma(\alpha) = \tau(\beta),$$

$$Id_\tau^\sigma(\alpha, \beta) = FALSE \Leftrightarrow \sigma(\alpha) \neq \tau(\beta).$$

It turns out that the latter criterion is stronger.

Theorem 3.3 ([63]) *If representations (S, σ) and (T, τ) are strongly isomorphic, then they are isomorphic.*

Proof.

Suppose that representations (S, σ) and (T, τ) are strongly isomorphic. We want to show that they are isomorphic. We shall show that there exists a computable translation F_τ^σ from (S, σ) to (T, τ) (the proof of existence of a translation in the opposite direction is analogous).

Let $\alpha \in S$. Since T is computable, let us consider a computable enumeration of all of its elements (without repetitions). We check all the numerals from T one by one until we find such $\beta \in T$ that $Id_\tau^\sigma(\alpha, \beta) = TRUE$ (we can check it because we assumed that these representations are strongly isomorphic). Then the translation returns β.

■

It turns out that the implication in the opposite direction does not hold (Corollary 3.6).

Theorem 3.4 ([63]) *For any representations (S, σ) and (T, τ) of Z with computable function $\chi_=$, these representations are strongly isomorphic if and only if they are isomorphic.*

Proof.

The implication ($\Rightarrow$) has already been proved for all representations. We shall now prove the implication ($\Leftarrow$). Suppose that (S, σ) and (T, τ) are isomorphic. Let F_τ^σ be a computable translation from (S, σ) to (T, τ). Let

$\alpha \in S$ and $\beta \in T$. We would like to know if $\sigma(\alpha) = \tau(\beta)$. We calculate $\beta' = F^\sigma_\tau(\alpha)$. Since $\chi_=$ is computable in (T, τ), we can calculate $Id^\sigma_\tau(\alpha, \beta)$ in the following way:

$$Id^\sigma_\tau(\alpha, \beta) = \chi_=(\beta, \beta').$$

■

This theorem is another example of two different notions becoming equivalent if we assume computability of characteristic function of identity. The previous example was computability of f and χ_f, for any function f (Theorem 2.10).

Theorem 3.5 ([63]) *If $\chi_=$ is not computable in (S, σ), then (S, σ) is not strongly isomorphic to any representation (not even to itself).*

Proof.

Let (S, σ) be a representation in which $\chi_=$ is not computable. Let (T, τ) be any representation. Suppose for the sake of contradiction that these two representations are strongly isomorphic. Then the following functions are computable: Id^σ_τ and, by Theorem 3.3, translations between these representations F^σ_τ and F^τ_σ. We will show how to compute $\chi_=$ in (S, σ). Let $\alpha, \beta \in S$. Note that we can calculate $F^\sigma_\tau(\beta)$ and $Id^\sigma_\tau(\alpha, F^\sigma_\tau(\beta))$.

The following conditions are equivalent:

1. $Id^\sigma_\tau(\alpha, F^\sigma_\tau(\beta)) = TRUE$,
2. $\sigma(\alpha) = \tau(F^\sigma_\tau(\beta))$,
3. $\alpha =^\sigma \beta$,
4. $\chi_=(\alpha, \beta) = TRUE$.

This, however, contradicts our assumption that $\chi_=$ is not computable.

■

Corollary 3.6 ([63]) *There exist representations (S, σ) and (T, τ) that are isomorphic, but not strongly isomorphic.*

Strong isomorphism is a very powerful notion indeed. It preserves all the properties of representations that are of interest to us here. In case of

representations without computable $\chi_=$, all relevant properties are preserved by isomorphism.[1]

Considering how powerful this criterion is, the theorem below provides us with a very useful tool. From the practical perspective, it means that any theorem that can be proved for all unambiguous representations also holds for every representation with computable $\chi_=$.

Theorem 3.7 ([63]) *For any representation (S, σ) of Z with computable function $\chi_=$ there exists an unambiguous representation (T, τ) of Z strongly isomorphic to (S, σ).*

Proof.

Let (S, σ) be a representation. Let $\alpha_0, \alpha_1, \alpha_2, ...$ be a computable enumeration of all numerals from S.

We define an infinite sequence of sets $T_0, T_1, T_2, ...$ as follows:

$$T_0 = \{\alpha_0\},$$

$$T_{n+1} = \begin{cases} T_n & \text{if } \exists_{i \leq n}\ \alpha_i =^\sigma \alpha_{n+1}, \\ T_n \cup \{\alpha_{n+1}\} & \text{otherwise.} \end{cases}$$

Let $T = \bigcup_{n \in \mathbb{N}} T_n$ and $\tau = \sigma \cap (T \times Z)$. We shall prove that (T, τ) is an unambiguous representation of Z strongly isomorphic to (S, σ).

First we shall prove that (T, τ) is a representation of Z. To this end we shall prove that all conditions from the definition of a representation hold.

1. $T \subseteq \Sigma^*$, where Σ is finite. That is the case because $T \subseteq S \subseteq \Sigma^*$, where Σ is finite.

2. T is computable. Let $\alpha \in \Sigma^*$. We want to check if $\alpha \in T$. We can check if $\alpha \in S$.

   If $\alpha \notin S$, then $\alpha \notin T$ because $T \subseteq S$.

   If $\alpha \in S$, then there exists $n \in \mathbb{N}$ such that $\alpha = \alpha_n$. Let us check numerals in the sequence $\{\alpha_n\}_{n \in \mathbb{N}}$ one by one until we find such n. Then:

   $$\alpha_n \in T \Leftrightarrow \neg \exists_{i<n} \alpha_i =^\sigma \alpha_n.$$

[1] Some ideas related to Turing reducibility discussed in section 7.2 are an exception, though.

We can check if this condition is satisfied because we only need to check for finitely many numerals if $\alpha_i =^\sigma \alpha_n$ holds. This is computable because we assumed that $\chi_=$ is computable in (S, σ).

Thus T is computable.

3. $\tau : T \to Z$ is a function onto Z. It is obvious that $\tau = \sigma \cap (T \times Z)$ is a function. We want to show that this function is onto Z.

   Let $a \in Z$. Since σ is onto Z we conclude that for a certain natural number n the following holds: $\sigma(\alpha_n) = a$.

   If $\alpha_n \in T$, then $\tau(\alpha_n) = a$.

   Otherwise there exists $i < n$ such that $\alpha_i =^\sigma \alpha_n$. Let us take the smallest such i. From the definition it follows that $\alpha_i \in T_i \subseteq T$. Therefore, we conclude that $\alpha_i \in T$ and $\tau(\alpha_i) = a$. It follows that τ is onto Z.

Therefore (T, τ) is a representation of Z.

It is a simple conclusion from the definition of the sequence $\{T_n\}_{n\in\mathbb{N}}$ that this representation must be unambiguous.

It is also strongly isomorphic to (S, σ). For let us take any $\alpha \in S$ and $\beta \in T \subseteq S$. Since $T \subseteq S$, it follows that:

$$Id^\sigma_\tau(\alpha, \beta) = \chi_=(\alpha, \beta).$$

However, we assumed that $\chi_=$ is computable in (S, σ).

■

3.2 Equivalence and χ-equivalence

Equivalence and χ-equivalence are further criteria we are going to consider. This time, instead of focusing on numerals and their intertranslatability between representations, we compare what functions and sets are computable in given representations.

Definition 3.8 *Representations (S, σ) and (T, τ) of Z are equivalent if and only if for any function f the following condition holds:*

f is computable in $(S, \sigma) \Leftrightarrow$ f is computable in (T, τ).

Definition 3.9 *Representations (S, σ) and (T, τ) of Z are χ-equivalent if and only if for any $R \subseteq Z^n$ the following condition holds:*

χ_R is computable in $(S, \sigma) \Leftrightarrow \chi_R$ is computable in (T, τ).

Theorem 3.10 ([63]) *If (S, σ) and (T, τ) are isomorphic representations of Z, then they are equivalent.*

Proof.

Let (S, σ) and (T, τ) be isomorphic representations of Z. Let F_τ^σ be a translation from (S, σ) to (T, τ) and F_σ^τ — a translation from (T, τ) to (S, σ). Let $f : Z \to Z$ be computable in (S, σ) (without loss of generality we assume that f is unary). We shall prove that f is computable in (T, τ).

Let $\beta \in T$. We calculate $f(\tau(\beta))$ in (T, τ). We know that there exists $\alpha \in S$ such that $F_\sigma^\tau(\beta) = \alpha$ and it follows from our assumption that we can find such α. Let $\alpha' = f^\sigma(\alpha)$, where f^σ is a computable function which represents f in (S, σ), and let $\beta' = F_\tau^\sigma(\alpha')$ (both these functions are computable due to our assumptions).

Note that the above algorithm guarantees that $f(\tau(\beta)) = \tau(\beta')$ for each $\beta \in T$. This is because $\sigma(\alpha) = \tau(\beta)$ and $\sigma(\alpha') = \tau(\beta')$ (by the definition of a translation) and $f^\sigma(\alpha) = \alpha'$. Hence f is computable in (T, τ).

The proof of the implication in the opposite direction is analogous.

■

Theorem 3.11 ([63]) *If (S, σ) and (T, τ) are isomorphic representations of Z, then they are χ-equivalent.*

Proof.

Let representations (S, σ) and (T, τ) be isomorphic and let $R \subseteq Z^n$ be such that χ_R is computable in (S, σ). We shall prove that χ_R is also computable in (T, τ).

Since both representations are isomorphic, there exists a computable translation $F_\sigma^\tau : T \to S$. Then for any $\alpha_1, ..., \alpha_n \in T$ the following holds:

$$\chi_R^\tau(\alpha_1, ..., \alpha_n) = \chi_R^\sigma(F_\sigma^\tau(\alpha_1), ..., F_\sigma^\tau(\alpha_n)),$$

where χ_R^σ and χ_R^τ are functions representing χ_R respectively in (S, σ) and (T, τ). Since χ_R^σ and F_σ^τ are computable, we conclude that χ_R^τ is also computable. ■

Theorem 3.12 *If (S, σ) and (T, τ) are equivalent representations of Z with computable $\chi_=$, then they are isomorphic.*

Proof.

Let $(\alpha_n)_{n\in\mathbb{N}}$ and $(\beta_n)_{n\in\mathbb{N}}$ be recursive enumerations without repetitions of all numerals in (S, σ) and (T, τ) respectively. We can assume without loss of generality that both these representations are unambiguous because every representation with computable function $\chi_=$ is strongly isomorphic to an unambiguous representation and strong isomorphism preserves all properties that concern us here.

Let f be the function computed in (S, σ) as follows:

$$f^\sigma(\alpha_n) = \alpha_{n+1}.$$

Since (S, σ) is unambiguous, this is certainly a well-defined function. Since (S, σ) and (T, τ) are equivalent, f is also computable in (T, τ). Let $\beta_k \in T$ be such that $\tau(\beta_k) = \sigma(\alpha_0)$. Then for any numeral $\alpha_i \in S$ we compute its translation as follows:

$$F^\sigma_\tau = (\alpha_i) = \underbrace{f^\tau ... f^\tau}_{i \text{ times}}(\beta_k).$$

Using an analogous argument, we show that there exists a computable translation F^τ_σ. As a result, these representations are isomorphic.

■

Theorem 3.13 *If (S, σ) and (T, τ) are χ-equivalent representations of Z with computable $\chi_=$, then they are isomorphic.*

Proof.

We utilise a very similar argument as in the proof of Theorem 3.12. We construct the function f defined as in that proof. Since f is computable in (S, σ), it follows from Theorem 2.10 that χ_f is also computable in (S, σ).

Since (S, σ) and (T, τ) are χ-equivalent, χ_f is computable in (T, τ), and — by Theorem 2.10 — so is f. We define a translation F^σ_τ in the same way as above.

■

Theorem 3.14 *For any representations (S, σ) and (T, τ) of Z with computable $\chi_=$ the following conditions are equivalent:*

1. (S, σ) *and* (T, τ) *are strongly isomorphic,*

2. (S, σ) *and* (T, τ) *are isomorphic*

3. (S, σ) *and* (T, τ) *are equivalent,*

4. (S, σ) *and* (T, τ) *are* χ*-equivalent.*

Proof.
It is a conclusion from Theorems 3.4, 3.10, 3.11, 3.12 and 3.13.

■

We now wish to show that the above theorem would not be valid without the assumption that $\chi_=$ is computable. We are going to construct two representations of $\mathbb{N}$ which are χ-equivalent but not equivalent or isomorphic.

Definition 3.15 *Let the representation* (S, σ) *of* $\mathbb{N}$ *be defined as follows:*

The alphabet Σ *consists of standard digits* $\overline{0}, ..., \overline{9}$*, a left bracket (, a right bracket) and a comma. The set of numerals* S *consists of all inscriptions of the form* $(\overline{a}, \overline{b})$*, where* $\overline{a}$*,* $\overline{b}$ *are standard numerals.*

Let $B \subseteq \mathbb{N}$ *be a set not computable in the standard representation. We define* σ *as follows:*

$$\sigma((\overline{a}, \overline{b})) = \begin{cases} a & \text{if } b \notin B, \\ a+1 & \text{if } b \in B. \end{cases}$$

Lemma 3.16 (S, σ) *defined as above is a correct representation of* $\mathbb{N}$*.*

Proof.
The only condition that might not be obvious is that for every natural number n there is a numeral $(\overline{a}, \overline{b}) \in S$ representing n. Let $n \in \mathbb{N}$. Since B is not computable, it follows that $B \neq \mathbb{N}$. Let $b \in \mathbb{N} \setminus B$. Then $\sigma(\overline{n}, \overline{b}) = n$.

■

Lemma 3.17 *Let* $A \subseteq \mathbb{N}$*. Then* χ_A *is computable in* (S, σ) *if and only if* $A = \emptyset$ *or* $A = \mathbb{N}$*.*

Proof.
The implication ($\Leftarrow$) is obvious. We shall prove ($\Rightarrow$). Suppose that $A \neq \emptyset$, $A \neq \mathbb{N}$ and that χ_A is computable in (S, σ).

There must be a number $n \in \mathbb{N}$ such that $n \notin A$ and $n+1 \in A$ (or $n \in A$ and $n+1 \notin A$ — but this case is analogous). For any $a \in \mathbb{N}$ the following conditions are equivalent:

1. $a \notin B$,
2. $\sigma((\overline{n}, \overline{a})) = n$,
3. $\sigma((\overline{n}, \overline{a})) \notin A$,
4. $\chi_A^\sigma((\overline{n}, \overline{a})) = FALSE$.

Analogously, for any $a \in \mathbb{N}$ the following conditions are equivalent:

1. $a \in B$,
2. $\sigma((\overline{n}, \overline{a})) = n + 1$,
3. $\sigma((\overline{n}, \overline{a})) \in A$,
4. $\chi_A^\sigma((\overline{n}, \overline{a})) = TRUE$.

Therefore:

$$\forall_{a \in \mathbb{N}}(a \in B \Leftrightarrow \chi_A^\sigma((\overline{n}, \overline{a})) = TRUE),$$

$$\forall_{a \in \mathbb{N}}(a \notin B \Leftrightarrow \chi_A^\sigma((\overline{n}, \overline{a})) = FALSE).$$

Therefore B is computable in the standard representation which constitutes a contradiction with our assumption.

■

Lemma 3.18 *Let $R \subseteq \mathbb{N}^k$. Then χ_R is computable in (S, σ) if and only if $R = \emptyset$ or $R = \mathbb{N}^k$.*

Proof.

The implication ($\Leftarrow$) is obvious. We shall prove ($\Rightarrow$). Suppose that $R \neq \emptyset$, $R \neq \mathbb{N}^k$ and that χ_R is computable in (S, σ).

For any $(n_1, ..., n_k), (n'_1, ..., n'_k) \in \mathbb{N}^k$, we shall call them neighbouring elements if they differ only on one coordinate, and on this coordinate they differ only by 1, i.e. if there is $1 \leq i \leq k$ such that $n_i = n'_i + 1$ or $n'_i = n_i + 1$ and for all $1 \leq j \leq k$, if $j \neq i$, then $n_j = n'_j$.

If $R \neq \emptyset$ and $R \neq \mathbb{N}^k$, then there must obviously exist $(n_1, ..., n_k)$ and $(n'_1, ..., n'_k)$ — two neighbouring elements of $\mathbb{N}^k$ such that $(n_1, ..., n_k) \in R$ and $(n'_1, ..., n'_k) \notin R$. Without loss of generality we can assume that $n_1 = n'_1 + 1$, and that $n_j = n'_j$, for $1 < j \leq k$. Let us fix $n_2, ..., n_k$.

Let $C = \{a \in \mathbb{N} : (a, n_2, ..., n_k) \in R\}$. Since C is neither $\emptyset$, nor $\mathbb{N}$, it follows from Lemma 3.17 that C is not computable in (S, σ). Then R is not computable in (S, σ) either. Thus we have obtained a contradiction. Therefore the only relations computable in (S, σ) are $\emptyset$ and $\mathbb{N}^k$.

■

Theorem 3.19 *There exist representations (S, σ) and (T, τ) of $\mathbb{N}$ that are χ-equivalent but not equivalent.*

Proof.

Let (S, σ) be defined as above. We shall construct (T, τ) by modifying (S, σ) in the following way:

Let f be any unary function not computable in (S, σ). We construct the alphabet of (T, τ) by adding the symbol $\overline{f}$ to the alphabet of (S, σ).

We construct T as follows:

For every numeral α: if $\alpha \in S$, then $\alpha \in T$.

For every numeral α: if $\alpha \in T$, then $\overline{f}\alpha \in T$.

We construct τ as follows:

For every numeral α: if $\alpha \in S$, then $\tau(\alpha) = \sigma(\alpha)$.

For every numeral α: if $\tau(\overline{f}\alpha) = f(\tau(\alpha))$.

Obviously, f is computable in (T, τ) as follows:

$$f^\tau(\alpha) = \overline{f}\alpha.$$

But we assumed that f is not computable in (S, σ). Therefore (S, σ) and (T, τ) are not equivalent. However, we shall prove that they are χ-equivalent.

The only sets and relations computable in (S, σ) are $\emptyset$ and $\mathbb{N}^k$ and they are computable in (T, τ) as well.

Suppose that a set or a relation is computable in (T, τ). Since $S \subseteq T$ and the interpretation of all numerals from S is the same in both representations, it follows that this set or relation is also computable in (S, σ). Therefore (S, σ) and (T, τ) are χ-equivalent but not equivalent.

■

Theorem 3.20 *There exist representations (S, σ) and (T, τ) of $\mathbb{N}$ that are χ-equivalent but not isomorphic.*

Proof.

Suppose that all χ-equivalent representations are isomorphic. Then they would all need to be equivalent as well due to Theorem 3.10 and that contradicts Theorem 3.19.

■

In this chapter we have defined several criteria for similarity of representations. We have proved that they are all equivalent if we assume computability of $\chi_=$. However, without such assumption this is not always the case.

When it comes to criteria discussed in this chapter, it turns out that representations with computable $\chi_=$ have much neater properties than other representations. In the following chapter we shall observe an analogous phenomenon with respect to computability of functions.

4

Computability of functions in various representations of natural numbers

In this chapter we want to consider computability of various functions on natural numbers.

Some of the theorems presented below were either proved by Shapiro in [52] or are generalisations of his results. Whenever this is the case, we state it explicitly. The proofs of Shapiro's theorems are not presented here unless we have formulated them in a more general way. To avoid confusion, all theorems are going to be formulated in our terminology, even if it differs from that used by Shapiro. Theorems 4.22, 4.23 and 4.24 result from joint work with D. Kalociński.

Further results regarding this topic are included in Chapter 5, where we focus on computability of functions in representations of $\mathbb{N}$ with computable $\chi_=$. We also discuss there computability of permutations in various representations.

4.1 Functions computable in every representation

Definition 4.1 *Let $f : \mathbb{N} \to \mathbb{N}$.*
If $\exists_k \exists_m \forall_{n>m} f(n) = k$, then we shall say that f is almost constant.
If $\exists_m \forall_{n>m} f(n) = n$, then we shall say that f is almost identity.

Theorem 4.2 (Shapiro [52]) *The only unary functions computable in every unambiguous representation are almost constant and almost identity functions.*

We are not going to prove this theorem here. However, it is an immediate conclusion from Theorem 4.23.

It is worth noting that due to Theorem 3.7 this result, along with other Shapiro's theorems cited here, can be extended to all functions with computable $\chi_=$. However, in general case the above theorem is not true. It should be replaced with:

Theorem 4.3 *The only unary functions computable in every representation are constant and identity functions.*

Proof.
It is clear that all constant and identity functions are computable in every representation. We need to prove that only these functions are. Suppose f is neither constant nor identity. Thanks to the previous theorem, we only need to consider the following two cases:

1. f is almost constant but not constant,
2. f is almost identity but not identity.

Case 1 We shall construct a representation (S, σ) in which f is not computable. Let S be the standard set of numerals. Let k be the value of f for nearly all arguments and $a_0, ..., a_m$ be all arguments for which f takes values other than k.

Let $A \subseteq N$ be a set not computable in the standard representation. We shall construct σ as follows:

$$\sigma(\overline{0}) = k.$$

Let $b_0, b_1, b_2, ...$ be an enumeration of all numbers from $A \setminus \{0\}$ and $c_0, c_1, c_2, ...$ — an enumeration of all numbers from $(\mathbb{N} \setminus A) \setminus \{0\}$.

For $i \in \mathbb{N}$, to each of the numerals $\overline{b_i}$, function σ shall assign one of numbers $a_0, ..., a_m$ and to each of the numerals $\overline{c_i}$ it shall assign one of numbers from the set $\mathbb{N} \setminus \{a_0, ..., a_m\}$. It shall be done in such a way that function σ shall be surjective and for every positive natural number t: $\sigma(\overline{t}) \neq k$.

This is possible unless k is the only argument for which f assumes a value different from k. We shall deal with this case later.

Suppose that f is computable in (S, σ). We shall show that contrary to our assumption A needs to be computable in the standard representation.

The algorithm provided below only works for $n \neq k$ but that does not need to bother us since the answer for k can be given explicitly.

Let $n \in \mathbb{N} \setminus \{k\}$. We want to know whether $n \in A$. We calculate $f^\sigma(\overline{n})$. The following are equivalent:

1. $f^\sigma(\overline{n}) \neq \overline{0}$,
2. $f(\sigma(\overline{n})) \neq k$,
3. $\sigma(\overline{n}) \in \{a_0, a_1, ..., a_m\}$,
4. $\overline{n} \in \{\overline{b_i} : i \in \mathbb{N}\}$,
5. $n \in \{b_i : i \in \mathbb{N}\}$,
6. $n \in A$.

Analogously, we can prove that $n \notin A \Leftrightarrow f^\sigma(\overline{n}) = \overline{0}$ and we have obtained a contradiction.

To complete the proof of the first case, we need to consider f of the following form:

$$f(n) = \begin{cases} k & \text{if } n \neq k, \\ l & \text{if } n = k, \end{cases}$$

where $k \neq l$. This is because in such a case it is not possible to construct σ in the way described above.

Let $A \subseteq \mathbb{N}$ be a set of natural numbers not computable in the standard representation. Let $a_0, a_1, a_2, ...$ be an enumeration of all numbers from $A \setminus \{0\}$ and $b_0, b_1, b_2, ...$ — an enumeration of all numbers from $(\mathbb{N} \setminus A) \setminus \{0\}$.

We shall construct a representation (S, σ) in which f is not computable. Let S be the standard set of numerals. We shall construct σ as follows:

$$\sigma(\overline{0}) = l,$$

$$\sigma(\overline{a_i}) = k, \text{for all } i \in \mathbb{N},$$

and to all the numerals $\overline{b_0}, \overline{b_1}, \overline{b_2}, ...$ we shall assign all the numbers other than k and l.

Suppose that f is computable in (S, σ). We shall show that contrary to our assumption A needs to be computable in the standard representation.

The algorithm provided below works only for $n \neq 0$ but that does not need to bother us because we can give the answer for 0 explicitly.

Let $n \in \mathbb{N} \setminus \{0\}$. We want to know whether $n \in A$. We calculate $f^\sigma(\overline{n})$. The following are equivalent:

1. $f^\sigma(\overline{n}) = \overline{0}$,
2. $f(\sigma(\overline{n})) = l$,
3. $\sigma(\overline{n}) = k$,
4. $\overline{n} \in \{\overline{a_i} : i \in \mathbb{N}\}$,
5. $n \in \{a_i : i \in \mathbb{N}\}$,
6. $n \in A$.

Analogously, we can prove that $n \notin A \Leftrightarrow f^\sigma(\overline{n}) \neq \overline{0}$ and we have obtained a contradiction.

Case 2 Assume that f is almost identity but not identity. Let $A \subseteq \mathbb{N}$ be a set of natural numbers not computable in the standard representation. Let $a_0, a_1, a_2, ...$ be an enumeration of all numbers from A and $b_0, b_1, b_2, ...$ — an enumeration of all numbers from $\mathbb{N} \setminus A$.

We shall construct a representation (S, σ) in which f is not computable. S shall be the standard set of numerals. Let $c_0, c_1, ..., c_m$ be natural numbers such that $f(c_i) \neq c_i$ for $i = 0, ..., m$ and let them be the only natural numbers with such a property.

Now let us construct σ. To each of the numerals $\overline{a_i}$, σ shall assign one of the numbers c_i and to each of the numerals $\overline{b_i}$, σ shall assign one of the

other numbers. This shall be done in such a way that each natural number is assigned to at least one numeral.

Suppose for the sake of contradiction that f is computable in (S, σ). We shall provide an algorithm for A. Let $n \in \mathbb{N}$. We want to know whether $n \in A$. We calculate $f^\sigma(\overline{n})$. For any n, the following conditions are equivalent:

1. $f^\sigma(\overline{n}) \neq \overline{n}$,
2. $\sigma(\overline{n}) \in \{c_0, c_1, ..., c_m\}$,
3. $\overline{n} \in \{\overline{a_i} : i \in \mathbb{N}\}$,
4. $n \in \{a_i : i \in \mathbb{N}\}$,
5. $n \in A$.

Analogously, for every natural number n the following holds:

$$f^\sigma(\overline{n}) = \overline{n} \Leftrightarrow n \notin A.$$

Therefore we have obtained a contradiction.

It follows that the only functions computable in every representation are constant and identity functions.

■

Definition 4.4 *A function $f : \mathbb{N}^k \to \mathbb{N}$ shall be called a projection if there exists $i \in \{1, ..., k\}$ such that:*

$$\forall_{x_1} ... \forall_{x_k}\ f(x_1, ..., x_k) = x_i.$$

Theorem 4.5 *The only functions $f : \mathbb{N}^k \to \mathbb{N}$ computable in every representation are constant functions and projections.*

Proof.

The implication ($\Leftarrow$) is obvious.

We shall prove the implication ($\Rightarrow$) by induction over k. The previous theorem constitutes the base case for this induction. Now suppose that the only functions of k arguments computable in every representation are constant functions and projections. Let $f : \mathbb{N}^{k+1} \to \mathbb{N}$ be a function computable

in every representation. We wish to show that it is either constant or a projection. In this proof we shall utilise Lemmas 4.6, 4.7, 4.8 and 4.9, included below.

For every $1 \leq i \leq k+1$ and every $j \in \mathbb{N}$, let us define a function:

$$f_{i,j} : \mathbb{N}^k \to \mathbb{N}$$

such that for all $x_1...., x_{i-1}, x_{i+1}, ..., x_{k+1}$:

$$f_{i,j}(x_1...., x_{i-1}, x_{i+1}, ..., x_{k+1}) = f(x_1...., x_{i-1}, j, x_{i+1}, ..., x_{k+1}),$$

i.e. a function obtained by substituting the value j for the variable x_i in f.

All functions $f_{i,j}$ are computable in every representation because they are obtained by substituting a value for a variable in f, and f is computable in every representations. Therefore, by inductive assumption, each of them is either constant or a projection.

We want to show that f is either a constant function or a projection. We have two cases to consider.

Case 1 Suppose that among all functions $f_{i,j}$ there is at least one projection $f_{i_0,j_0} = x_l$. Then by Lemmas 4.6 and 4.7, every function $f_{i,j}$ is also a projection on the same coordinate x_l, unless $i = l$. Hence, for any $i \neq l$ and any $x_1, ..., x_{k+1}$:

$$f(x_1, ..., x_{k+1}) = f_{i,x_i}(x_1...., x_{i-1}, x_{i+1}, ..., x_{k+1}) = x_l.$$

Therefore, f is a projection on x_l.

Case 2 Suppose that all functions $f_{i,j}$ are constant. Then by Lemmas 4.8 and 4.9 all these functions are identical and always equal to the same value c. Hence f is also constant and equal to c.

■

Note that functions $f_{i,j}$ mentioned in subsequent lemmas are those defined in the proof of Theorem 4.5.

Lemma 4.6 *If f_{i_1,j_1} is a projection on x_l and $i_2 \neq l$, then f_{i_2,j_2} is also a projection.*

Proof.

Suppose to the contrary that the function f_{i_2,j_2} is not a projection, i.e. it is constant. Assume that for all arguments:

$$f_{i_1,j_1}(x_1, ..., x_{k+1}) = x_l$$

and

$$f_{i_2,j_2}(x_1, ..., x_{k+1}) = c.$$

Let us consider the following cases:

Case 1 Suppose that $i_1 \neq i_2$. We know that for every sequence of arguments:

$$f_{i_1,j_1}(x_1, ..., x_{i_1-1}, x_{i_1+1}, ..., x_{i_2-1}, x_{i_2}, x_{i_2+1}, ..., x_{k+1}) = x_l.$$

In particular, for $x_{i_2} = j_2$:

$$f_{i_1,j_1}(x_1, ..., x_{i_1-1}, x_{i_1+1}, ..., x_{i_2-1}, j_2, x_{i_2+1}, ..., x_{k+1}) = x_l.$$

But this is also the value of f_{i_2,j_2}, with j_1 substituted for x_{i_1}. This is however a contradiction since this is always equal to x_l, and we assumed that f_{i_2,j_2} is constant. Therefore, f_{i_2,j_2} must be a projection.

Case 2 Suppose that $i_1 = i_2$. If $j_1 = j_2$, then it is trivial. Hence suppose that $j_1 \neq j_2$.

Let $A \subseteq \mathbb{N}$ be uncomputable. We shall construct a representation (S, σ). Let S be the standard set of decimal numerals and let

$$\sigma(\overline{2n}) = \begin{cases} c & \text{if } n = 0, \\ j_1 & \text{if } n > 0 \wedge n \in A, \\ j_2 & \text{if } n > 0 \wedge n \notin A. \end{cases}$$

Assign the remaining numbers to numerals of the form $\overline{2n+1}$ in any injective way.

To obtain a contradiction, we want to construct an algorithm which decides whether $n \in A$. The answer for $n = 0$ is given explicitly as a special case. Assume $n > 0$. Since f is computable in every representation, we compute the value of f^σ, where we substitute the numeral $\overline{2n}$ for x_{i_1} (which is the same variable as x_{i_2}), the numeral $\overline{1}$ — for x_l, and for other variables we substitute any numerals.

Due to the construction of σ and because $n > 0$, the numeral substituted for x_{i_1} represents either j_1 or j_2. If it represents j_1, then f is a projection on x_l and it has to return a numeral which represents the same number as the numeral $\overline{1}$ — hence it has to return the numeral $\overline{1}$, since no other numeral represents the same number. If, on the other hand, the numeral substituted for x_{i_1} represents j_2, then f is a constant function always equal to c and in this case the algorithm returns the numeral $\overline{0}$.

Therefore, if the algorithm returns $\overline{1}$, then $n \in A$. If it returns $\overline{0}$, then $n \notin A$.

■

Lemma 4.7 *If f_{i_1,j_1} and f_{i_2,j_2} are both projections, then they are projections on the same coordinate.*

Proof.

Suppose that $f_{i_1,j_1} = x_{l_1}$ and $f_{i_2,j_2} = x_{l_2}$ are different projections, i.e. $l_1 \neq l_2$. Let us consider the following cases:

Case 1 Suppose that $i_1 \neq i_2$. We know that for every sequence of arguments:

$$f_{i_1,j_1}(x_1, ..., x_{i_1-1}, x_{i_1+1}, ..., x_{i_2-1}, x_{i_2}, x_{i_2+1}, ..., x_{k+1}) = x_{l_1}.$$

In particular, for $x_{i_2} = j_2$:

$$f_{i_1,j_1}(x_1, ..., x_{i_1-1}, x_{i_1+1}, ..., x_{i_2-1}, j_2, x_{i_2+1}, ..., x_{k+1}) = x_{l_1}.$$

But this is also equal to f_{i_2,j_2}, with j_1 substituted for x_{i_1}, hence it is always equal to x_{l_2}. This is a contradiction since we can substitute different values for x_{l_1} and x_{l_1}.

Case 2 Suppose that $i_1 = i_2$. If $j_1 = j_2$, then it is trivial. Hence suppose that $j_1 \neq j_2$. Let $A \subseteq \mathbb{N}$ be a set uncomputable in the standard representation. We are going to construct a representation (S, σ). Let S be the standard set of numerals and let:

$$\sigma(\overline{2n}) = \begin{cases} j_1 & \text{if } n \in A, \\ j_2 & \text{if } n \notin A. \end{cases}$$

Assign the rest of numbers to the remaining numerals in any injective way.

To obtain a contradiction, we shall construct an algorithm which decides whether $n \in A$. Since f is computable in every representation, we are going to compute f^σ, with $\overline{2n}$ substituted for x_{i_1} (which is equal to x_{i_2}), $\overline{1}$ — for x_{l_1} and $\overline{3}$ — for x_{l_2}. If the output numeral is $\overline{1}$, then the algorithm has computed projection on coordinate x_{l_1}. Then $\sigma(\overline{2n}) = j_1$ and $n \in A$. Analogously, if the output numeral is 3, then $n \notin A$. Hence A is computable in the standard representation and we have obtained a contradiction.

■

Lemma 4.8 *If $i_1 \neq i_2$ and functions $f_{i_1,j_1} = c_1$ and $f_{i_2,j_2} = c_2$ are constant, then $c_1 = c_2$.*

Proof.

Let these functions be constant and assume values, respectively, c_1 and c_2. Without loss of generality assume that $i_1 < i_2$. We shall show that $c_1 = c_2$. Then for any x_1, ... , x_{k+1}:

$$\begin{aligned} c_1 &= f_{i_1,j_1}(x_1, ..., x_{i_1-1}, x_{i_1+1}, ..., x_{i_2-1}, j_2, x_{i_2+1}, ..., x_{k+1}) \\ &= f(x_1, ..., x_{i_1-1}, j_1, x_{i_1+1}, ..., x_{i_2-1}, j_2, x_{i_2+1}, ..., x_{k+1}) \\ &= f_{i_2,j_2}(x_1, ..., x_{i_1-1}, j_1, x_{i_1+1}, ..., x_{i_2-1}, x_{i_2+1}, ..., x_{k+1}) \\ &= c_2. \end{aligned}$$

■

Lemma 4.9 *If $f_{i_0,j_0} = c$ is constant, then:*

1. *$f_{i_0,j} = c$, for every j, if all the functions $f_{i,j}$ are constant,*
2. *$f_{i_0,j} = j$, for every j, if at least one function $f_{i,j}$ is a projection.*

Proof.

First note that if $f_{i_0,j_0} = c$ is constant, then $f_{i_0,j}$ must be constant for every j. Otherwise $f_{i_0,j}$ would be a projection, for some j, and then, by Lemma 4.6, f_{i_0,j_0} would also be a projection. That would be a contradiction because f_{i_0,j_0} is constant.

Suppose that all the functions $f_{i,j}$ are constant. Consider the function f_{i_1,j_1} such that $i_1 \neq i_0$. By Lemma 4.8, it is also equal to c. Then, for any j we can again apply Lemma 4.8 to f_{i_1,j_1} and $f_{i_0,j}$ and we conclude that $f_{i_0,j} = c$, for every j.

Now suppose that the function f_{i_1,j_1} is a projection on x_l. Then, by Lemmas 4.6 and 4.7, all functions $f_{i,j}$ are projections on x_l unless $i = l$. Since all functions $f_{i_0,j}$ are constant, it follows that $l = i_0$. Then for all $i \neq i_0$ and all j, functions $f_{i,j}$ are projections on x_{i_0}, and for all j, functions $f_{i_0,j}$ are constant and equal to j.

■

Theorem 4.10 (Shapiro [52]) *The only subsets of $\mathbb{N}$ whose characteristic functions are computable in every unambiguous representation are finite and cofinite sets.*

Theorem 4.11 *The only sets of natural numbers whose characteristic functions are computable in every representation are* $\emptyset$ *and* $\mathbb{N}^k$*, for* $k \in \mathbb{N}$*.*

Proof.
The only sets whose characteristic functions are computable in the representation described in Definition 3.15 are $\emptyset$ and $\mathbb{N}^k$. This is due to Lemma 3.18.

■

Shapiro showed in [52] that there exist functions not computable in any unambiguous representation. However, it is worth pointing out that every function is computable in a certain (possibly ambiguous) representation.

Theorem 4.12 (Shapiro [52]) *There exists a function on natural numbers which is not computable in any unambiguous representation.*

Corollary 4.13 *Every function on natural numbers is computable in a certain representation.*

Proof.
This is an immediate conclusion from a later Theorem 4.26.

■

4.2 Computability of successor, addition, multiplication and exponentiation

In this section we are going to examine computability of functions on natural numbers, in particular of successor, addition, multiplication and exponentiation, and relations between them. We also show how the situation changes when computability of $\chi_=$ is additionally assumed.

Theorem 4.14 ([63]) *Let* (S, σ) *be a representation of* $\mathbb{N}$*. The following conditions are equivalent:*

1. (S, σ) *is isomorphic to the standard representation of* $\mathbb{N}$*.*
2. (S, σ) *is strongly isomorphic to the standard representation of* $\mathbb{N}$*.*
3. *The successor function and* $\chi_=$ *are computable in* (S, σ)*.*

Proof.

The equivalence of (1) and (2) follows from Theorem 3.4.

The implication (1) $\Rightarrow$ (3) follows from Theorems 3.10 and 3.11.

We shall now prove the implication (3) $\Rightarrow$ (1).

Let (S, σ) be a representation of $\mathbb{N}$ in which the successor function (denoted as $Succ$) and $\chi_=$ are computable. In this representation there is a numeral representing number 0. Let us denote such a numeral as λ.

We shall show how to compute translations between (S, σ) and the standard representation.

Let α be a numeral of (S, σ). For every natural number n, let us denote

$$\alpha_n = \underbrace{Succ^\sigma(Succ^\sigma(...Succ^\sigma(\lambda)...))}_{Succ \text{ iterated } n \text{ times}}.$$

We compare one by one each α_n with α until we find such n that $\sigma(\alpha) = \sigma(\alpha_n)$. Then $\sigma(\alpha) = n$, so the numeral $\overline{n}$ represents the same number in the standard representation as the numeral α in (S, σ).

In order to translate a standard numeral $\overline{n}$ to (S, σ), we calculate α_n defined as above.

■

The above theorem is a generalisation of Shapiro's result included in [52]. Since Shapiro only considered unambiguous representations, he did not need to worry about computability of $\chi=$ (which is trivial in unambiguous representations). Hence, he only demanded that successor should be computable in every representation in order for it to be 'acceptable' (this is discussed in more detail in section 8.2). However, with our broader definition of representation, this assumption is not sufficient by itself.

Theorem 4.15 ([63]) *There exists a representation (S, σ) of $\mathbb{N}$ in which the successor function is computable, but addition, multiplication and exponentiation are not computable.*

Proof.

We are going to construct such a representation (S, σ). The alphabet consists of symbols: $\overline{0}$, $\overline{1}$, a. The set of numerals S consists of all finite nonempty sequences of symbols from the alphabet which contain at most one occurrence of a. Obviously, S is computable.

Let $A \subseteq \mathbb{N}$ be uncomputable in the standard representation. We construct σ in the following way:

$$\sigma(\overline{0}) = 0,$$
$$\sigma(\overline{1}) = 1,$$
$$\sigma(a) = 0 \Leftrightarrow 1 \notin A,$$
$$\sigma(a) = 1 \Leftrightarrow 1 \in A.$$

Also, for any $\alpha \in S$:

$$\sigma(\alpha \frown \overline{0}) = \sigma(\alpha),$$
$$\sigma(\alpha \frown \overline{1}) = \sigma(\alpha) + 1,$$
$$\sigma(\alpha \frown a) = \sigma(\alpha) \Leftrightarrow lh(\alpha) + 1 \notin A,$$
$$\sigma(\alpha \frown a) = \sigma(\alpha) + 1 \Leftrightarrow lh(\alpha) + 1 \in A,$$

where $\frown$ is a concatenation and $lh(\alpha)$ is the length of the sequence α.

This is a correct representation because every natural number n is represented by at least one numeral, namely $\overline{1}...\overline{1}$ consisting of n digits $\overline{1}$, with the exception of number 0, which is represented by the numeral $\overline{0}$.

A representation of the successor function in (S, σ) can be defined as follows:

$$Succ(\alpha) = \alpha \frown \overline{1}.$$

This function is computable.

We shall show that addition is not computable in this representation. Suppose to the contrary that it is.

For any natural number $n \geq 1$ let us denote:

$$\lambda_n = \overline{0}...\overline{0}a,$$

where λ_n consists of $n-1$ digits $\overline{0}$ followed by one occurrence of a.

For any sequence $\alpha \in S$, let $\#_{\overline{1}}(\alpha)$ and $\#_a(\alpha)$ denote respectively the number of occurrences of symbols $\overline{1}$ and a in the numeral α.

We want to find out whether $n \in A$. We compute $\lambda_n + \lambda_n$ in (S, σ). We know that $\sigma(\lambda_n)$ is equal to 0 or 1. Thus $\sigma(\lambda_n + \lambda_n)$ is equal to 0 or 2. If $n \in A$, then $\sigma(\lambda_n) = 1$ and $\sigma(\lambda_n + \lambda_n) = 2$. Then $_{\overline{1}}(\lambda_n + \lambda_n) \geq 1$. If, however, $n \notin A$, then $\sigma(\lambda_n) = \sigma(\lambda_n + \lambda_n) = 0$ and then $\#_{\overline{1}}(\lambda_n + \lambda_n) = 0$.

It is easy to find out which of these cases occurs and thus — whether $n \in A$. It follows that A is computable in the standard representation, which contradicts our assumption. Therefore, addition is not computable in (S, σ).

Similarly we show that multiplication and exponentiation are not computable in (S, σ). Let us denote:

$$\delta_n = \overline{1}...\overline{1}a,$$

where λ_n consists of $n-1$ digits $\overline{1}$ followed by one occurrence of a. Then we compute respectively $\delta_n \cdot \delta_n$ or ${\delta_n}^{\overline{11}}$ in (S, σ) (note that they both return the same result, we shall only provide a proof for the case with multiplication).

Assume that multiplication is computable in (S, σ). We shall prove that A is also computable then. Let $n \in \mathbb{N}$. We want to find out whether $n \in A$. Without loss of generality we can assume that $n \geq 2$.[1] Let $\alpha \in S$ be the result of multiplication $\delta_n \cdot \delta_n$ in (S, σ). We know that $\sigma(\delta_n)$ is equal to either $n-1$ or n. Therefore:

1. If $\sigma(\delta_n) = n-1$, then $\sigma(\delta_n \cdot^\sigma \delta_n) = (n-1)^2 = n^2 - 2n + 1$. Therefore $\#_{\overline{1}}(\alpha) = n^2 - 2n$ or $\#_{\overline{1}}(\alpha) = n^2 - 2n + 1$.

2. If $\sigma(\delta_n) = n$, then $\sigma(\delta_n \cdot^\sigma \delta_n) = n^2$. Therefore $\#_{\overline{1}}(\alpha) = n^2 - 1$ or $\#_{\overline{1}}(\alpha) = n^2$.

Note that for $n \geq 2$ we can find out which of these cases occurs. If the first case occurs, then $n \notin A$, otherwise $n \in A$. Thus we have obtained contradiction with the assumption that A is not computable. Therefore multiplication (and similarly exponentiation) is not computable in (S, σ).

■

We are going to show now that computability of addition implies computability of successor, but not of multiplication or exponentiation.

Theorem 4.16 ([64]) *Let (S, σ) be a representation of $\mathbb{N}$ in which addition is computable. Then the successor function is also computable in this representation.*

Proof.

Let (S, σ) be a representation of $\mathbb{N}$ in which addition is computable. In (S, σ) there must be a numeral representing number 1. Let us denote this numeral as β. Then we can calculate the successor function in (N, σ) as follows:

$$Succ^\sigma(\alpha) = \alpha +^\sigma \beta.$$

■

[1] The algorithm which is supposed to find out whether $n \in A$ will have answers for $n \in \{0, 1\}$ explicitly given as special cases.

Theorem 4.17 ([64]) *There exists a representation (S, σ) of $\mathbb{N}$ in which addition (and hence also successor) is computable, but multiplication and exponentiation are not computable.*

Proof.

For any natural number n, let $\overline{n}$ denote the numeral which represents n in the standard representation of $\mathbb{N}$.

We construct the following representation (S, σ):

The alphabet Σ consists of digits $\overline{0}$, ... , $\overline{9}$, of symbols (,) and the comma.

We construct the set S of numerals as follows:

For any standard numerals $\overline{a_0}$, ... , $\overline{a_n}$, the sequence $(\overline{a_0}, ..., \overline{a_n})$ is a numeral of the representation (S, σ) if $a_0 \geq \sum_{i=1}^{n} a_i$. Clearly, S is computable.

Let $A \subseteq \mathbb{N}$ be uncomputable in the standard representation and such that $0 \in A$.

For any $(\overline{a_0}, ..., \overline{a_n}) \in S$ the function σ is defined as follows:

$$\sigma((\overline{a_0}, ..., \overline{a_n})) = \sum_{i=0}^{n} (a_i \cdot \chi_A(i)).$$

This representation is well-defined because every natural number is represented by at least one numeral, in particular n is represented by $(\overline{n})$.

For any numerals $(\overline{a_0}, ..., \overline{a_k})$ and $(\overline{b_0}, ..., \overline{b_l})$ (without loss of generality we can assume that $k \leq l$), we define addition in (S, σ) in the following way:

$$(\overline{a_0}, ..., \overline{a_k}) +^{\sigma} (\overline{b_0}, ..., \overline{b_l}) = (\overline{a_0 + b_0}, ..., \overline{a_k + b_k}, \overline{b_{k+1}}, ...\overline{b_l}).$$

It is obviously computable. It can be easily shown that this is indeed a correct definition of addition in (S, σ).

We shall prove that multiplication is not computable in this representation. Suppose that it is computable. We shall show that then A is computable in the standard representation which leads to a contradiction.

We want to find out whether $n \in A$.

For any natural number n we define the following numeral:

$$\lambda_n = (\overline{1}, \overline{0}, ..., \overline{0}, \overline{1}),$$

where λ_n has $\overline{1}$ on the 0th and n-th position and $\overline{0}$ on all the other positions.

We compute the multiplication $\lambda_n \cdot \lambda_n$ in (S, σ). There are two possible cases:

If $n \in A$, then $\sigma(\lambda_n) = 2$ and $\sigma(\lambda_n \cdot \lambda_n) = 4$. From the condition that $a_0 \geq \sum_{i=1}^{n} a_i$ it follows that $a_0 \geq 2$ for every numeral representing number 4 in this representation.

If $n \notin A$, then $\sigma(\lambda_n) = 1$ and $\sigma(\lambda_n \cdot \lambda_n) = 1$, so $a_0 = 1$ in a numeral representing number 1 in this representation.

We determine which of these cases occurs and thus we can find out whether $n \in A$. Therefore A is computable in the standard representation, which leads to a contradiction. It follows that multiplication is not computable in (S, σ).

Similarly, by considering the result of the computation ${\lambda_n}^{\lambda_n}$ we can show that exponentiation is not computable in this representation.

We compute ${\lambda_n}^{\lambda_n}$ in (S, σ). There are two possible cases:

If $n \in A$, then $\sigma(\lambda_n) = 2$ and $\sigma({\lambda_n}^{\lambda_n}) = 4$. From the condition that $a_0 \geq \sum_{i=1}^{n} a_i$ it follows that $a_0 \geq 2$ for every numeral representing number 4 in this representation.

If $n \notin A$, then $\sigma(\lambda_n) = 1$ and $\sigma({\lambda_n}^{\lambda_n}) = 1$, so $a_0 = 1$ in a numeral representing number 1 in this representation.

We determine which of the cases occurs and thus we can find out whether $n \in A$. Therefore A is computable in the standard representation, which leads to a contradiction. It follows that exponentiation is not computable in (S, σ).

■

Note that computability of addition and $\chi_=$ implies that it is (strongly) isomorphic to the standard one. This conclusion follows from Theorems 4.14 and 4.16.

However, surprisingly, computability of multiplication even in tandem with $\chi_=$ does not ensure computability of any of the other functions under consideration.

Theorem 4.18 ([64]) *There exists a representation (S, σ) of $\mathbb{N}$ in which multiplication and $\chi_=$ are computable, but addition and exponentiation are not computable.*

Proof.

Let π be a permutation of $\mathbb{N}$ uncomputable in the standard representation.

We construct the following representation. The alphabet consists of the digits $\overline{0}$, ... , $\overline{9}$, of the symbols (,) and the comma.

The set S consists of $\overline{0}$ and all finite sequences of the form $(\overline{a_0}, ..., \overline{a_n})$ (where each a_i is a natural number).

For any numerals $(\overline{a_0}, ..., \overline{a_k})$ and $(\overline{b_0}, ..., \overline{b_l})$ (without loss of generality we assume that $k \leq l$), we define multiplication in (S, σ) in the following way:

$$(\overline{a_0}, ..., \overline{a_k}) \cdot^{\sigma} (\overline{b_0}, ..., \overline{b_l}) = (\overline{a_0 + b_0}, ..., \overline{a_k + b_k}, \overline{b_{k+1}}, ...\overline{b_l}),$$

and the product of two numerals is equal to $\overline{0}$ if at least one of these numerals is $\overline{0}$.

We construct σ as follows:

$$\sigma((\overline{a_0}, ..., \overline{a_n})) = p_{\pi(0)}^{a_0} \cdot ... \cdot p_{\pi(n)}^{a_n}$$

(we adopt the convention that p_i is the i-th prime number).

As a consequence of the fundamental theorem of arithmetic, (S, σ) is a valid representation of $\mathbb{N}$ and $\chi_=$ is computable in it. Multiplication is computable as well, since it can be calculated by adding respective elements of two sequences to each other, according to the formula given above. We shall show that addition and exponentiation are not computable in this representation.

Let us assume that addition is computable in this representation. We shall show that π must be computable in the standard representation, which leads to a contradiction.

Let n be any natural number. We want to find the value of $\pi^{-1}(n)$. We take any non-zero numeral $\lambda \in S$ and we calculate $\underbrace{\lambda + ... + \lambda}_{p_n \text{ times}}$ in (S, σ). Then we check on which position of λ the number has increased by 1. The number of this position is equal to $\pi^{-1}(n)$. Thus we can compute the permutation π^{-1} in the standard representations. However, if π^{-1} is computable, then obviously π is also computable.

Now suppose that exponentiation is computable in this representation. We shall prove that then the permutation π must be computable in the standard representation.

For any natural number n we shall find $\pi(n)$ using the following method:

Let λ_n be a numeral of the form $(\overline{0}, ..., \overline{0}, \overline{1})$, where the digit $\overline{1}$ is proceeded by n occurrences of the digit $\overline{0}$. Then $\sigma(\lambda_n) = p_{\pi(n)}$. We compute the result of $(\overline{1})^{\lambda_n}$ in (S, σ). Obviously:

$$\sigma((\overline{1})^{\lambda_n}) = p_{\pi(0)}^{p_{\pi(n)}}.$$

When we calculate this exponentiation, we will get the numeral $(\overline{p_{\pi(n)}})$ as a result. Thus we find out the value of the $\pi(n)$-th prime number, so we can easily compute $\pi(n)$.

■

If we assume computability of exponentiation, then — just like in the case of successor — computability of the remaining functions follows only if we additionally assume computability of $\chi_=$. We prove this in the following two theorems.

Theorem 4.19 ([64]) *There exists a representation (S, σ) of $\mathbb{N}$ in which exponentiation is computable, but successor, addition and multiplication are not computable.*

Proof.

We construct such a representation as follows:

The alphabet consists of digits $\overline{0}$, ... , $\overline{9}$, symbols $\overline{\pi}$, E, (,) and the comma.

The set of numerals is the smallest set S satisfying the following conditions:

Every numeral of the standard representation belongs to S.

If $\alpha, \beta \in S \setminus \{\overline{0}, \overline{1}\}$, then $E(\alpha, \beta) \in S$.

If $\alpha \in S$ and α represents a prime number in the standard representation, then $\overline{\pi}(\alpha) \in S$.

Let π be a permutation of prime numbers (i.e. a bijection from prime numbers onto prime numbers) uncomputable in the standard representation, such that $\pi(2) = 2$. We construct σ in the following way:

For any standard numeral $\overline{n}$, let $\sigma(\overline{n}) = n$.

For any $\alpha, \beta \in S \setminus \{\overline{0}, \overline{1}\}$, let $\sigma(E(\alpha, \beta)) = \sigma(\alpha)^{\sigma(\beta)}$.

For any prime number p, let $\sigma(\overline{\pi}(\overline{p})) = \pi(p)$.

This representation is well-defined because every natural number is represented by a certain numeral, in particular by the same numeral as in the standard representation.

We define exponentiation in (S, σ) as follows:

$\alpha^\beta = E(\alpha, \beta)$, for $\alpha, \beta \in S \setminus \{\overline{0}, \overline{1}\}$,

$\alpha^{\overline{0}} = \overline{1}$, $\alpha^{\overline{1}} = \alpha$, $\overline{1}^\alpha = \overline{1}$, for any $\alpha \in S$,

$\overline{0}^\alpha = \overline{0}$, for any $\alpha \in S \setminus \{\overline{0}, \overline{1}\}$.

Exponentiation is computable in this representation.

We shall prove that successor is not computable in (S, σ). Suppose to the contrary that it is computable. We shall show that π is then computable in the standard representation, which leads to a contradiction.

Let $T = (\alpha_{ij})_{i,j \in \mathbb{N}}$ be defined as follows:

$$\alpha_{ij} = Succ^{\sigma}(Succ^{\sigma}(...(\overline{\pi}(\overline{p_i}))...)),$$

where the successor is iterated j times, and p_i is the i-th prime number.

Since we assumed that successor is computable, it follows that T is a computable family of numerals indexed by pairs of natural numbers.

Note that each prime number p is represented by exactly two numerals in (S, σ), namely $\overline{p}$ and $\overline{\pi}(\overline{q})$, for a certain prime number q. Let us consider the following cases:

Case 1 Suppose that in a certain row of T there is no numeral of the form $\overline{p}$ representing a prime number, i.e. all prime numbers in this row are represented by numerals of the form $\overline{\pi}(\overline{p})$. Let i be such that for all $j \in \mathbb{N}$, $\alpha_{i,j} = \overline{\pi}(\overline{p_n})$, where all p_n are prime numbers. Since this row contains representations of nearly all (i.e. all but finitely many) prime numbers, let P be the set of all prime numbers whose representations are not in this row.

We are going to construct an algorithm which computes $\pi(n)$, for any $n \in \mathbb{N}$. For all such n that $\pi(n) \in P$, answers are given explicitly as special cases. For all the other cases, let j be such that

$$\alpha_{i,j} = \overline{\pi}(\overline{p_n}).$$

Then:

$$\pi(p_n) = \pi(p_i) + j.$$

Note that this is computable because p_i is a fixed number, independent of n.

Case 2 Suppose that in every row of T there is a numeral of form $\overline{p}$ representing a certain prime number. Let n be any natural number. We shall show how to compute $\pi(p_n)$, where p_n is the n-th prime number. Let j, p be such that $\alpha_{n,j} = \overline{p}$, where p is a prime number. Then:

$$\pi(p_n) + j = p.$$

It follows that:

$$\pi(p_n) = p - j.$$

We have obtained a contradiction with the assumption that π is not computable in the standard representation. Therefore the successor function is not computable in (S, σ).

From this and Theorem 4.16 it follows that addition is not computable in (S, σ) either.

We shall show that multiplication is not computable in (S, σ). Assume to the contrary that it is. Let $p > 2$ be a prime number. We shall show how to compute $\pi(p)$.

Let us calculate $\overline{2} \cdot^{\sigma} \overline{\pi}(\overline{p})$. From the definition of (S, σ) it follows that the result of this calculation cannot be of the form $E(\alpha, \beta)$ for any numerals α, β. It cannot be of the form $\overline{\pi}(\overline{q})$, for any prime number q, either, because the result of this multiplication is not a prime number. Therefore it must be a certain standard numeral $\overline{n}$. However, all such numerals are interpreted in (S, σ) just like in the standard representation. Therefore $\pi(p) = \frac{n}{2}$.

It follows that π is computable in the standard representation, which contradicts our assumption. Therefore, multiplication is not computable in (S, σ).

■

Theorem 4.20 ([64]) *Let (S, σ) be a representation of $\mathbb{N}$ in which exponentiation and $\chi_=$ are computable. Then multiplication and of addition are also computable in this representation.*

Proof.

Let $\alpha, \beta \in S$. We want to calculate the values of $\alpha \cdot^{\sigma} \beta$ and $\alpha +^{\sigma} \beta$. Let λ be a numeral representing number 2 in (S, σ) and let $(\zeta_n)_{n \in \mathbb{N}}$ be a recursive enumeration of S. For each ζ_n, we check in (S, σ) if the following equality holds:

$$(\lambda^{\alpha})^{\beta} = \lambda^{\zeta_n}$$

until we find a numeral for which it is true. Such ζ_n shall be the result of calculating $\alpha \cdot \beta$ in (S, σ).

To calculate $\alpha + \beta$ in (S, σ), we check for each ζ_n whether the following equality holds:

$$\lambda^{\alpha} \cdot^{\sigma} \lambda^{\beta} = \lambda^{\zeta_n}.$$

until we find a numeral for which it is true. Such ζ_n shall be the result of calculating $\alpha + \beta$ in (S, σ).

We conclude that addition and multiplication are computable in (S, σ).

■

The following theorem is meant to show that assuming computability of successor or addition together with $\chi=$ is not the only way of guaranteeing

that a representation must be isomorphic to the standard one. A different, unintuitive assumption about representation is just as good. We shall later refer to this theorem in our philosophical discussion contained in section 8.2.

Theorem 4.21 *Let (S, σ) be a representation of $\mathbb{N}$. Then the following conditions are equivalent:*

1. *(S, σ) is isomorphic to the standard representation.*
2. *$\chi_{\leq}$ is computable in (S, σ) and there exists $f : \mathbb{N} \to \mathbb{N}$ — a strictly increasing (but not identity) function computable both in (S, σ) and in the standard representation.*

Proof.

The implication (1) $\Rightarrow$ (2) is obvious from Theorems 3.10 and 3.11. We need to prove the other implication.

From the computability of $\chi_{\leq}$, it follows that $\chi_{=}$ is computable in (S, σ) as well. Due to Theorem 3.7 we can assume without loss of generality that (S, σ) is unambiguous.

Let n be the least such number that $n \neq f(n)$. For $i = 0, ..., n-1$ let α_i be such that $\sigma(\alpha_i) = i$. For $i \geq n$ we continue constructing the sequence of numerals $\{\alpha_i\}_{i \in \mathbb{N}}$ as follows:

$$\alpha_{i+1} = f^{\sigma}(\alpha_i).$$

This sequence is computable because f^{σ} is computable. Since it is strictly increasing for $i \geq n$, it holds that $f(i) > i$ for all $i \geq n$. Therefore the sequence thus defined is strictly increasing to infinity.

Since f is also computable in the standard representation, a translation of all numerals from $\{\alpha_i\}_{i \in \mathbb{N}}$ to the standard representation is also computable.

We know that S is computably enumerable. Let $\{\zeta_i\}_{i \in \mathbb{N}}$ be a computable enumeration of all numerals from S. We shall provide an algorithm which computes $F^{\sigma}_{\sigma_{st}}$ — a translation between (S, σ) and the standard representation. Let $\beta \in S$. We want to find $F^{\sigma}_{\sigma_{st}}(\beta)$.

Let $L = (\alpha_0)$ be a finite sequence of numerals of length 1, with α_0 defined as above. We shall expand and rearrange this sequence in stages $s = 0, 1, 2, ...$ Suppose that s stages have already been performed. In stage $s + 1$, we add the numeral ζ_{s+1} to L, unless it was already there, and we rearrange elements of L in accordance with the ordering of numbers they

represent. This can be done because we assumed that $\chi_<$ is computable in (S, σ).

At a certain stage, β will be added to this sequence. Let i be such that $\sigma(\alpha_i) > \sigma(\beta)$ (such α_i certainly exists and will be added to L at a certain stage of the algorithm). Let $m = \sigma(\alpha_i)$. We are able to compute m and we know that exactly m numerals will be added to L in front of α_i (even if we do not know when they will be added). Therefore, we continue adding numerals to L until there are exactly m numerals in front of α_i. We know that these numerals represent numbers from 0 to $m - 1$ in their standard ordering. Then, based on the position of β in the sequence, we can determine what number it stands for.

■

4.3 Computability of sets and relations

Theorem 4.22 *There exists a representation (S, σ) of $\mathbb{N}$ in which $\chi_<$ is computable, but the successor function is not computable.*

Proof.

We begin with an informal description of the construction. The idea is that we are going to construct an infinite computable sequence of numerals (without repetitions). The position of each numeral α in the sequence will determine its denotation $\sigma(\alpha)$. This will suffice to ensure that $\chi_<$ is computable. The construction of the sequence should guarantee that no algorithm computes the successor. Since each algorithm computing some unary partial function from numerals to numerals is a potential candidate for computing the successor function, we need to ensure that for each algorithm e there is such numeral α that if e stops with α as its input, then its output β satisfies the following condition:

$$\sigma(\alpha) + 1 \neq \sigma(\beta).$$

This will be done by assigning to each algorithm e a distinct numeral and waiting for it to stop (with α as its input) in order to perform appropriate manipulations on the sequence to ensure that the sequence satisfies the condition stated above. The details are included below.

Construction Let S be an infinite computable set of numerals over a finite alphabet and $(\zeta_n)_{n\in\mathbb{N}}$ — a computable enumeration of S. Let $(e_i)_{i\in\mathbb{N}}$ be a computable enumeration of all algorithms, each computing some partial function from S to S.

In the 0th stage of the construction let $A_0 = (\zeta_0)$ and $L_0 = ((e_0, \zeta_0))$ (it means that we shall run the 0th algorithm on numeral ζ_0).

Suppose we have already performed n stages of the construction. Let $A_n = (\alpha_0, \alpha_1, ..., \alpha_k)$ and let $L_n = ((e_0, \alpha_{i_0}), (e_1, \alpha_{i_1}), ..., (e_m, \alpha_{i_m}))$, where $0 \leq i_0, i_1, ..., i_m \leq k$.

There are two possible cases:

1. At least one algorithm e_l stops with its assigned numeral α_{i_l} supplied as the input in at most n steps.

2. None of the algorithms $e_0, ..., e_m$ stops in at most n steps with its assigned numeral supplied as the input.

In the first case, let e_l be an algorithm which stops with the corresponding numeral supplied as its input. Then let β be the numeral returned as the output and let $\beta' = \zeta_t$, where t is the least index not utilised in $A_n \cup \{\beta\}$. Since it is our aim to ensure that the algorithm under consideration is not an algorithm for a successor, we do one of the following depending on the situation:

1. if β is already in A_n and it appears on this list directly behind α, then let $A_{n+1} = (\alpha_0, ..., \alpha, \beta', \beta, ..., \alpha_k)$,

2. if β is already in A_n and but it does not appear on this list directly behind α, then let $A_{n+1} = A_n$,

3. if β does not appear in A_n and α is the last element there, then let $A_{n+1} = (\alpha_0, ..., \alpha_k, \beta', \beta)$,

4. if β does not appear in A_n and α is not the last element there, then let $A_{n+1} = (\alpha_0, ..., \alpha_k, \beta)$,

After that, let $L_{n+1} = L_n \setminus \{(e_l, \alpha)\}$. The above procedure needs to be performed for each such e_l.

In the second case, let β be the numeral with the least index not utilised in A_n and let e be the unused algorithm with the smallest index. Then add β at the end of A_n and (e, β) at the end of L_n. Thus we have constructed A_{n+1} and L_{n+1}. This ends the construction.

Verification Let:

$$A = \lim_{n\to\infty} A_n.$$

The limit in the above definition is understood in the following way:

$$\lim_{n\to\infty} A_n = A \Leftrightarrow \ \forall_i \, \exists_{n_0} \forall_{n>n_0} \, A_n(i) = A(i),$$

where $A_n(i)$ and $A(i)$ denote the i-th element of A_n and A, respectively.

We define σ as follows: $\sigma(A(i)) = i$, for all $i \in \mathbb{N}$.

This is a representation of $\mathbb{N}$.

The set of numerals S is computable by definition. Every numeral from S is included in the sequence A because the second of the two cases happens infinitely many times. Suppose to the contrary that it happens only finitely many times. Then after a certain point, in each step we would always need to follow instructions for the first case which include, among others, removing a certain element from L_n. It follows that after finitely many steps L_n would be empty and in the following step we would need to follow instructions from the second case (no algorithm would be able to stop because the list of algorithms would be empty). This leads to a contradiction.

We also need to show that A is a well-defined sequence of numerals, i.e. that for each numeral in this sequence there are only finitely many numerals in front of it. That is the case because we always add new numerals at the end of the sequence unless an algorithm e with input α stops and returns β as the output while β has already been on the list A_n directly behind α. However, only finitely many numerals can be added this way to the sequence in front of any given numeral. This is because every numeral can be assigned to only one algorithm and no algorithm is assigned to new numerals added to this sequence via method described in the first case.

In (S, σ) the ordering is computable because we are able to generate approximations of A by performing the algorithm described above. We have also ensured that once two numerals appear on the list, their order is never reversed. Thus, in order to compare numbers represented by two numerals, we perform the algorithm until both these numerals appear on the list and then we check which of them comes earlier.

However, the successor function is not computable in (S, σ) because by construction of A we have ensured that none of the algorithms computes this function.

■

This theorem can be strengthened.

Theorem 4.23 *For every computable $f : \mathbb{N} \to \mathbb{N}$ which is neither almost constant, nor almost identity, there exists (S, σ) — a representation of $\mathbb{N}$ such that $\chi_<$ is computable in (S, σ), but f is not computable.*

Proof.

Let $f : \mathbb{N} \to \mathbb{N}$ be a computable function that is neither almost constant nor almost identity.

The idea of construction used here is similar to the one utilised in the proof of the previous theorem. We construct a computable infinite sequence of numerals (without repetitions), each denoting (via σ) its position in the sequence. Thus we ensure that $\chi_<$ is computable in this representation. Along the way, we discard each algorithm that could possibly compute f by performing appropriate manipulations on the sequence if an algorithm stops with the numeral assigned to it supplied as the input. The details of the construction are presented below.

Let S be an infinite computable set of numerals over a finite alphabet and $(\zeta_n)_{n\in\mathbb{N}}$ — a computable enumeration of S. Let $(e_i)_{i\in\mathbb{N}}$ be a computable enumeration of all algorithms, each computing some partial function from S to S.

We are going to ensure that none of the algorithms e_i computes f in (S, σ). To achieve this, we are going to construct a list of conditions that need to be satisfied at each step of the algorithm.

Construction In the 0th stage of the construction let $A_0 = (\zeta_0)$ and $L_0 = ((e_0, \zeta_0))$ (it means that we shall run the 0th algorithm with the numeral ζ_0 taken as input). The list of conditions is empty at the beginning.

Suppose we have already performed n stages of the construction. Let $A_n = (\alpha_0, \alpha_1, ..., \alpha_k)$ and let $L_n = ((e_0, \alpha_{i_0}), (e_1, \alpha_{i_1}), ..., (e_m, \alpha_{i_m}))$, where $0 \leq i_0, i_1, ..., i_m \leq k$.

There are two possible cases:

1. At least one algorithm e_l stops with its assigned numeral α_{i_l} taken as input in at most n steps.

2. None of the algorithms stops in at most n steps with its assigned numeral taken as input.

In the first case, for every (e_l, α) from the list such that an algorithm stops with its corresponding numeral as the input, let β be the numeral returned as the output.

Also, for each new algorithm that stops, add the following condition to the list:

$$f^{\sigma}(\alpha) \neq \beta.$$

We say that numerals α and β appear in this condition.

After that, remove each such (e_l, α) from L_n.

We are going to ensure that all conditions from the list are satisfied in each step. Every time we do this, we go through all numerals in A_n, one by one, in the same order in which they are situated there. If numerals α and β appear in a certain condition, we modify the sequence A_n only when we reach the later one of them (according to their position in A_n). We shall satisfy each condition by adding new numerals to the sequence (if necessary), either behind both α and β, or directly in front of the one that comes later. This will help us avoid a situation in which we again spoil a condition which has already been satisfied.

For each of the conditions, we ensure that it is satisfied in the following way:

1. if $\alpha = \beta$ is on the i-th position, then we need to find such $i' > k$ that $f(i') \neq i'$. Then move α to such position and fill all positions between i and i' with unused numerals. This is possible since f is not an almost identity function.

2. if α is in in A_n on the i-th position, β is already on the j-th position and $i < j$, then add new numerals directly in front of β to push β to such position $j' > k$ that $f(i) \neq j'$. This is possible since f is not an almost constant function.

3. if α is in in A_n on the i-th position, β is already on the j-th position and $i > j$, then add new numerals directly in front of α to push α to such position $i' > k$ that $f(i') \neq j$. This is possible since f is not an almost constant function.

4. if α is in in A_n on the i-th position and β does not appear in A_n at all, then put β on such position $j > k$ that $f(i) \neq j$ and fill all empty positions between α and β with unused numerals. This is possible since f is not an almost constant function.

Once all conditions have been satisfied, we have obtained a new sequence which shall be called A_{n+1}.

In the second case, let β be the numeral with the smallest index not utilised in A_n and let e be the unused algorithm with the smallest index. Add β at the end of A_n and (e, β) at the end of the L_n. Thus we have constructed A_{n+1} and L_{n+1}. This ends the construction.

Verification Let:

$$A = \lim_{n \to \infty} A_n.$$

The limit in the above definition is understood as in the previous proof. We define σ as follows: $\sigma(A(i)) = i$, for all $i \in \mathbb{N}$. This is a representation of $\mathbb{N}$.

The set of numerals S is computable by definition. Every numeral from S is included in the sequence A because in each step we add at least one numeral to the sequence.

Also, A is a well-defined sequence of numerals, i.e. that for each numeral in this sequence there are only finitely many numerals in front of it. That is the case because, as we are going to show, all numerals from each initial segment of sequence A have their positions fixed after a certain number of steps.

Note that each numeral is added to the sequence A either as the input, or the output of a certain algorithm, or as a 'filler' numeral put between other numerals to push them to certain positions within the sequence. We want to show that, once a numeral λ has been added to the sequence, only finitely many numerals can later be added in front of λ.

Let λ' be a numeral added to A when λ is already there. If λ' is added as the input or the output of a certain algorithm, then it must be added behind λ. The only situation when we need to add new numerals in front of λ is when they are filler numerals — but for each λ this can only be the case finitely many times. Note that whenever we add such filler numerals with α as the input and β as the output, we do so either behind both α and β or directly in front of the one which comes later (in terms of position in the sequence). Hence we only need to add filler numerals in front of λ if there is α in front of λ which was added to the sequence prior to λ and was assigned as the input to one of the algorithms. But there can only be finitely many such numerals α.

Therefore, at a certain stage of the construction, each numeral occupies a fixed position in the sequence which does not change at any later stage. This also guarantees that all conditions from the list are satisfied in (S, σ).

The ordering is computable in (S, σ) because we are able to generate approximations of A by performing the algorithm described above. We have also ensured that once two numerals appear on the list, their order is never reversed. Thus, in order to compare numbers represented by two numerals, we perform the algorithm until both these numerals appear on the list and then we check which of them comes earlier.

However, f is not computable in (S, σ) because by construction of A we have ensured that none of the algorithms computes this function.

■

Note that in every representation of $\mathbb{N}$ the computability of each of the following functions: $\chi_<$, $\chi_>$, $\chi_\leq$, $\chi_\geq$ is equivalent, and each of them implies that $\chi_=$ is computable. Hence Theorems 4.22 and 4.23 are equally valid if $\chi_<$ is replaced with any of these relations.

This is another theorem proved using a similar method:

Theorem 4.24 *There exists a representation (S, σ) of $\mathbb{N}$ in which $\chi_<$ is computable, but the characteristic function of divisibility $\chi_|$ is not computable.*

Proof.

Let S be a computable set of numerals and $(\zeta_n)_{n\in\mathbb{N}}$ — a computable enumeration of all elements of S without repetitions. Let $(e_n)_{n\in\mathbb{N}}$ be a computable enumeration of all algorithms on S which, given two numerals as input, return a logical value TRUE or FALSE as output. We shall provide an algorithm for the construction of an infinite sequence of numerals from S. During this construction, to each index of an algorithm we shall assign a pair of numerals on which this algorithm is going to be performed. The limit sequence obtained after infinitely many steps of this construction shall contain exactly one occurrence of every numeral from S. To each numeral we shall assign a natural number — the order of numbers will be the same as the order of numerals in the sequence.

Construction In the 0th stage of the construction let $A_0 = (\zeta_0, \zeta_1, \zeta_2, \zeta_3)$ and let $L_0 = ((e_0, \zeta_2, \zeta_3))$ (it means that we shall run the 0th algorithm on a pair of numerals ζ_2 and ζ_3). We are also going to construct a list of conditions which need to be satisfied at every stage. This list is empty at the beginning.

Suppose that we have already performed the first n stages of the construction, $n \geq 0$. Let $A_n = (\alpha_0, \alpha_1, ..., \alpha_k)$ and let

$$L_n = ((e_0, \alpha_{i_0}, \beta_{i_0}), (e_1, \alpha_{i_1}, \beta_{i_1}), ..., (e_m, \alpha_{i_m}, \beta_{i_m})),$$

where $0 \leq i_0, i_1, ..., i_m \leq k$.

In the $n+1$th stage do the following (in order to ensure that none of the algorithms computes characteristic function of divisibility):

1. Take an unused (i.e. one that has never been in L_i for any $i \leq n$) algorithm e with the smallest index and take two unused numerals with the least indices λ_1 and λ_2 and assign these numerals to that algorithm.

2. Let A_{n+1} be equal to A_n with λ_1 and λ_2 added at the end.

3. Let L_{n+1} be equal to L_n with $(e, \lambda_1, \lambda_2)$ added at the end.

4. Perform the first $n+1$ steps of all algorithms from L_{n+1} with their respective numerals as the input.

5. For each algorithm e_l working on numerals α_{i_l} and β_{i_l} that stops during these steps:

   (a) If e_l returns $TRUE$, then do nothing (because for sufficiently large numbers, it cannot be the case that one of the subsequent natural numbers is a divisor and the other one is its multiplicity).

   (b) If e_l returns $FALSE$, then we add $\alpha_{i_l} | \beta_{i_l}$ to the list of conditions.

   Then remove $(e_l, \alpha_{i_l}, \beta i_l)$ from L_{n+1}.

Now (if it is necessary) we are going to modify A_{n+1} in such a way as to ensure that each condition from the list is satisfied. To achieve this, we go through all numerals in A_{n+1}, one by one, in the same order in which they are situated there. If numerals α_{i_l} and β_{i_l} appear in a certain condition, we modify the sequence A_{n+1} when we reach β_{i_l}. The modification consists in adding sufficiently many new numerals directly in front of β_{i_l} to push it to a position in the sequence whose index is a multiple of the index of the position of α_{i_l}. Note that due to the construction of A_{n+1}, β_{i_l} always appears behind α_{i_l}. The placement of new filler numerals described here will help us avoid a situation in which we we spoil a condition which has already been satisfied.

This ends the construction.

Verification This procedure generates a sequence containing all numerals because in each step we add to it two numeral with least unused indices and the procedure consists of infinitely many steps. Also, no repetitions of numerals are possible because in each step only unused numerals are added.

This is a well-defined sequence, since position of each numeral in A is fixed at a certain stage of the construction. We are going to prove this now. Suppose that a numeral λ has been added to the sequence at a certain stage. Note that all numerals added later are going to be placed behind λ, except for those added to satisfy one of the conditions from the list. This, however, can only occur finitely many times, and each time we add finitely many filler numerals. This is because all numerals added to A_n this way must be placed directly in front of a numeral β_{i_l} assigned to a certain algorithm as its second argument and it is possible only if β_{i_l} was added to A_n before λ was. Therefore A is a well-defined sequence.

It follows from the above that all conditions from the list are going to be satisfied in A.

To compare numbers represented by two numerals, we perform the algorithm described above until both numbers are in the sequence and then check which of them comes first. This is a correct method because once two numerals have been added to the sequence in a specific order, their order cannot be reversed (even if at a later step some additional numerals are added between them).

■

Theorem 4.25 *Let F be a non-empty finite family of subsets of $\mathbb{N}$ and A be a subset of $\mathbb{N}$. Then the following conditions are equivalent:*

1. *for every (S, σ) — representation of $\mathbb{N}$: if for every $B \in F$, χ_B is computable in (S, σ), then χ_A is also computable in (S, σ),*
2. *A is a Boolean combination of sets from F.*

Proof.

The implication (2) $\Rightarrow$ (1) is obvious. We need to prove the other implication.

Suppose that A is not a Boolean combination of sets from F. We shall construct a representation (S, σ) for which the condition from (1) does not hold. Suppose that there are s sets in F. Let:

$$\varphi : \mathbb{N} \to \{TRUE, FALSE\}^n$$

be defined in the following way:

$$\varphi(n) = (\chi_{B_1}(n), ..., \chi_{B_s}(n)),$$

where B_1, ... , B_s are all the sets from F. Let us consider equivalence classes of the following equivalence relation defined on $\mathbb{N}$:

$$n_1 \equiv_\varphi n_2 \Leftrightarrow \varphi(n_1) = \varphi(n_2).$$

There are finitely many (at most 2^s) such equivalence classes. Let us enumerate them: E_0, ... , E_{k-1}.

We want to construct (S, σ). Let S be the standard set of numerals. For every $i < k$, σ shall assign numbers from E_i to numerals of the form $\overline{i + tk}$, for all natural numbers $t > 0$, i.e. numerals: $\overline{i}$, $\overline{i + k}$, $\overline{i + 2k}$, ... This assignment does not have to be injective. We shall formulate an additional constraint on σ later.

Since we assumed that A is not a Boolean combination of any sets from F, it cannot be equal to a sum of any equivalence classes of $\equiv_\varphi$. Therefore, there is an equivalence class E_m and numbers $m_1, m_2 \in E_m$ such that $m_1 \in A$ and $m_2 \notin A$. We have already decided that all numbers from E_m must be represented by numerals of the form $\overline{m + tk}$, for $t \in \mathbb{N}$. For any natural number $m < k$, let $N_m = \{\overline{m + tk} : t \in \mathbb{N}\}$. Now we shall formulate an additional constraint on how to assign numbers from E_m to numerals from N_m. Let $C \subseteq \mathbb{N}$ be a set not computable in the standard representation. Let:

$$\sigma(\overline{m + 2tk}) = \begin{cases} m_1 & \text{if } t \in C, \\ m_2 & \text{if } t \notin C. \end{cases}$$

The remaining numbers from E_m can be assigned to numerals from N_m in any way.

We shall show that χ_A is not computable in (S, σ). Suppose to the contrary that it is. Then for any natural number r:

$$\chi_C(r) = TRUE \Leftrightarrow \sigma(\overline{m + 2rk}) = m_1 \Leftrightarrow \chi_A^\sigma(\overline{m + 2rk}) = TRUE$$

and

$$\chi_C(r) = FALSE \Leftrightarrow \sigma(\overline{m + 2rk}) = m_2 \Leftrightarrow \chi_A^\sigma(\overline{m + 2rk}) = FALSE.$$

Therefore:

$$\chi_C(r) = \chi_A^\sigma(\overline{m + 2rk})$$

is computable (m and k are fixed numbers). Thus we have obtained a contradiction.

We assumed that (2) does not hold and constructed a representation (S, σ) such that for every $B \in F$, χ_B is computable in (S, σ), but χ_A is not. Therefore (1) $\Rightarrow$ (2).

■

Theorem 4.26 ([63]) *For any countable (finite or infinite) set of functions on natural numbers F and any $A \subseteq \mathbb{N}$ such that $A \neq \emptyset$, $A \neq \mathbb{N}$, there exists a representation (S, σ) of $\mathbb{N}$ in which all functions from F are computable, but χ_A is not computable.*

Proof.

Without loss of generality let us assume that $F = \{f_i\}_{i \in \mathbb{N}}$ is infinite. For every natural number n, let $\overline{n}$ be the numeral representing n in the standard representation.

We construct (S, σ) as follows:

The alphabet Σ consists of standard digits $\overline{0}$, ... , $\overline{9}$, symbol $\overline{f}$, symbols (,) and the comma.

All numerals of the standard representation are also numerals of (S, σ). Additionally, for any $\alpha_1, ..., \alpha_n \in S$ and any function $f_i \in F$, let:

$$\overline{f}\ \underbrace{\overline{1}...\overline{1}}_{i \text{ times}}(\alpha_1, ..., \alpha_n) \in S.$$

Let $B \subseteq \mathbb{N}$ be any set not computable in the standard representation. Both B and $\mathbb{N} \setminus B$ must be infinite then.

Function σ assigns a number to each standard numeral $\overline{n}$ in such a way that:

$$\sigma(\overline{n}) \in A \Leftrightarrow n \in B.$$

and every natural number is represented by at least one standard numeral. Such assignment exists because A and $\mathbb{N} \setminus A$ are non-empty while B and $\mathbb{N} \setminus B$ are infinite.

Also, for any natural number i and any $\alpha \in S$, let:

$$\sigma(\overline{f}\ \underbrace{\overline{1}...\overline{1}}_{i \text{ times}}(\alpha)) = f_i(\sigma(\alpha)).$$

This representation is well-defined because each natural number is represented by at least one numeral, namely a certain numeral of the standard representation.

For any function $f_i \in F$ and any $\alpha_1, ..., \alpha_n \in S$, if $f_i : \mathbb{N}^k \to \mathbb{N}$ and $k = n$, then let:

$$(f_i)^\sigma(\alpha_1, ..., \alpha_n) = \overline{f}\ \underbrace{\overline{1}...\overline{1}}_{i \text{ times}}(\alpha_1, ..., \alpha_n).$$

Otherwise, i.e. if $k \neq n$, we can assign any value to such a numeral.

It follows that all the functions from F are computable in (S, σ).

We shall show that χ_A is not computable in (S, σ). From the construction of σ it follows that for every natural number n:

$$n \in B \Leftrightarrow \chi_A^\sigma(\overline{n}) = TRUE.$$

If χ_A were computable in (S, σ), then B would be computable in the standard representation, contrary to the assumption. Therefore χ_A is not computable in (S, σ).

■

5

Normal representations and generating permutations

5.1 Representations generated by permutations

In this section we are going to consider representations of $\mathbb{N}$ with the standard set of numerals N and with computable $\chi_=$. We want to find out what functions are computable in them and what is their relation to the standard representation. To this aim, we shall introduce notions of normal representations and generating permutations. Theorems proved here are going to be useful in further considerations regarding computability of functions.

Definition 5.1 *A representation (S, σ) of $\mathbb{N}$ is normal if it is unambiguous and S is equal to the standard set of numerals N.*

Theorem 5.2 *For any representation (S, σ) of $\mathbb{N}$ with computable function $\chi_=$, there exists a normal representation (N, τ) of $\mathbb{N}$ strongly isomorphic to (S, σ).*

Proof.

From Theorem 3.7 it follows that (S, σ) is strongly isomorphic to an unambiguous representation. Let $(\alpha_n)_{n\in\mathbb{N}}$ be a computable enumeration of

all numerals of such an unambiguous representation. We construct (N, τ) by replacing every α_n with $\overline{n}$.

■

Definition 5.3 *Let (S, σ) be a representation of $\mathbb{N}$ with computable $\chi_=$ and let π be a permutation of $\mathbb{N}$. We say that (S, σ) is generated by π if (S, σ) is (strongly) isomorphic to the normal representation (N, σ_π) such that for every $n \in \mathbb{N}$:*

$$\sigma_\pi(\overline{n}) = \pi(n).$$

(N, σ_π) shall be called the canonical representation generated by π.

Corollary 5.4 *Let (S, σ) be a representation of $\mathbb{N}$ and π — a permutation of $\mathbb{N}$. Then (S, σ) is generated by π if and only if it is (strongly) isomorphic to the canonical representation generated by π.*

Definition 5.5 *The relation $\equiv_P$ on the set of all permutations of $\mathbb{N}$ shall be defined as follows:*

$$\pi_1 \equiv_P \pi_2 \Leftrightarrow \pi_2^{-1} \circ \pi_1 \text{ is computable (in the standard representation).}$$

Corollary 5.6 *$\equiv_P$ is an equivalence relation on the set of all permutations of $\mathbb{N}$.*

Proof.

It is reflexive because $\pi^{-1} \circ \pi = id$ for every permutation π and id is computable.

It is symmetrical because if $\pi_2^{-1} \circ \pi_1$ is computable, then

$$\pi_1^{-1} \circ \pi_2 = (\pi_2^{-1} \circ \pi_1)^{-1}$$

is also computable.

It is transitive because if both $P_1 = \pi_2^{-1} \circ \pi_1$ and $P_2 = \pi_3^{-1} \circ \pi_2$ are computable, then

$$\pi_3^{-1} \circ \pi_1 = P_2 \circ P_1$$

is also computable.

■

Theorem 5.7 *For any permutations π_1 and π_2 of $\mathbb{N}$, if $\pi_1 \equiv_P \pi_2$ then $\pi_1 \equiv_T \pi_2$ (i.e π_1 and π_2 are Turing-equivalent).*

Proof.

Let us assume that $\pi_1 \equiv_P \pi_2$. Therefore $\pi_2^{-1} \circ \pi_1$ is computable. We shall show that $\pi_1 \leq_T \pi_2$ (the proof in the opposite direction is analogous). We want to find the value of $\pi_1(n)$ assuming we can determine the value of $\pi_2(m)$ for any natural number m. We can do this by computing:

$$\pi_1(n) = (\pi_2 \circ (\pi_2^{-1} \circ \pi_1))(n).$$

■

Theorem 5.8 *If (S, σ) and (T, τ) are representations of $\mathbb{N}$ with computable $\chi_=$, π_1 and π_2 are permutations of $\mathbb{N}$ and:*

1. *(S, σ) is generated by π_1,*
2. *$(S, \sigma) \cong (T, \tau)$,*
3. *$\pi_1 \equiv_P \pi_2$,*

then (T, τ) is generated by π_2.

Proof.

Assume that $(S, \sigma) \cong (T, \tau) \cong (N, \nu_1)$, where (N, ν_1) is the canonical representation generated by π_1. This is possible due to Corollary 5.4. Note that isomorphism of representations is transitive and this is a property which we utilise in this proof.

Since (N, ν_1) is the canonical representation generated by π_1 and

$$(T, \tau) \cong (N, \nu_1),$$

it follows from Corollary 5.4 that (T, τ) is generated by π_1. We want to show that it is also generated by π_2. To this end we shall show that it is strongly isomorphic to (N, ν_2) — the canonical representation generated by π_2. We shall show that $(N, \nu_1) \cong (N, \nu_2)$.

We are looking for a computable translation $F_{\nu_2}^{\nu_1}$ such that:

$$F_{\nu_2}^{\nu_1}(\overline{m}) = \overline{n} \Leftrightarrow \pi_1(m) = \pi_2(n).$$

Since $\pi_1 \equiv_P \pi_2$, it means that $\pi_2^{-1} \circ \pi_1$ is computable. Let $n \in \mathbb{N}$. Then

$$F_{\nu_2}^{\nu_1}(\overline{m}) = \overline{n} \Leftrightarrow n = (\pi_2^{-1} \circ \pi_1)(m).$$

Hence $F_{\nu_2}^{\nu_1}$ is computable. A translation in the opposite direction can be constructed in an analogous way, considering that $\equiv_P$ is symmetrical. Therefore (T, τ) is generated by π_2.

■

Theorem 5.9 *If (S, σ) and (T, τ) are representations of $\mathbb{N}$ generated respectively by permutations π_1 and π_2, then the following conditions are equivalent:*

1. $(S, \sigma) \cong (T, \tau)$,
2. $\pi_1 \equiv_P \pi_2$.

Proof.

Let (S, σ) and (T, τ) be representations of $\mathbb{N}$ generated respectively by permutations π_1 and π_2.

We shall prove the implication (1) $\Rightarrow$ (2).

Let us assume that $(S, \sigma) \cong (T, \tau)$. Let (N, ν_1) and (N, ν_2) be canonical representations generated respectively by π_1 and π_2. Then obviously

$$(S, \sigma) \cong (N, \nu_1)$$

and

$$(T, \tau) \cong (N, \nu_2).$$

It follows that

$$(N, \nu_1) \cong (N, \nu_2)$$

and hence there exists a computable translation $F_{\nu_2}^{\nu_1}$ such that:

$$F_{\nu_2}^{\nu_1}(\overline{m}) = \overline{n} \Leftrightarrow \pi_1(m) = \pi_2(n) \Leftrightarrow n = (\pi_2^{-1} \circ \pi_1)(m),$$

since for every natural number m: $\nu_1(\overline{m}) = \pi_1(m)$ and $\nu_2(\overline{m}) = \pi_2(m)$. Hence $\pi_2^{-1} \circ \pi_1$ is computable. Therefore $\pi_1 \equiv_P \pi_2$.

We shall prove the implication (2) $\Rightarrow$ (1).

Let us assume that $\pi_1 \equiv_P \pi_2$. Hence $\pi_2^{-1} \circ \pi_1$ is computable. Let (N, ν_1) and (N, ν_2) be canonical representations generated respectively by π_1 and π_2. Then $(S, \sigma) \cong (N, \nu_1)$ and $(N, \nu_2) \cong (T, \tau)$. Let $F_{\nu_1}^{\sigma}$ and $F_{\tau}^{\nu_2}$ be respective computable translations. For any $\alpha \in S$:

$$F_{\tau}^{\sigma}(\alpha) = (F_{\tau}^{\nu_2} \circ F_{\nu_2}^{\nu_1} \circ F_{\nu_1}^{\sigma})(\alpha),$$

where F_τ^σ and $F_{\nu_2}^{\nu_1}$ are respective computable translations. Also, $F_{\nu_2}^{\nu_1}$ is computable because for all natural numbers m, n:

$$F_{\nu_2}^{\nu_1}(\overline{m}) = \overline{n} \Leftrightarrow \pi_1(m) = \pi_2(n) \Leftrightarrow n = (\pi_2^{-1} \circ \pi_1)(m).$$

Therefore $(N, \nu_1) \cong (N, \nu_2)$ and as a consequence $(S, \sigma) \cong (T, \tau)$.

■

Theorem 5.10 *Every representation of $\mathbb{N}$ with computable $\chi_=$ is generated by countably infinitely many permutations and all of these permutations are equivalent with respect to $\equiv_P$.*

Proof.

Let (S, σ) be generated by π. Let $(\pi_n)_{n\in\mathbb{N}}$ be an enumeration without repetitions of all computable permutations (in the standard representation). Let $(\pi'_n)_{n\in\mathbb{N}}$ be such that for any natural number n, $\pi_n = \pi'_n \circ \pi$. Obviously such a sequence $(\pi'_n)_{n\in\mathbb{N}}$ exists and is uniquely determined by the enumeration $(\pi_n)_{n\in\mathbb{N}}$.

Let us observe that $(\pi'_n)_{n\in\mathbb{N}}$ is the enumeration of all permutations equivalent to π with respect to $\equiv_P$. If $\pi_i \neq \pi_j$, then $\pi'_i \neq \pi'_j$, hence there are no repetitions in this sequence. Therefore this sequence consists of infinitely countably many permutations. Furthermore. each of these permutations generates (S, σ) due to Theorem 5.8.

■

Shapiro proved in [52] that there exists an unambiguous representation in which a function $f : \mathbb{N} \to \mathbb{N}$ is computable if and only if $f = \pi \circ g \circ \pi^{-1}$ for a certain function g computable in the standard representation and a certain π — permutation of $\mathbb{N}$. We are going to prove a stronger version of his result which ties it to the concept of generating permutations.

Theorem 5.11 *If (S, σ) is a representation of $\mathbb{N}$ with computable $\chi_=$ generated by π, then for any $f : \mathbb{N}^k \to \mathbb{N}$ the following conditions are equivalent:*

1. *there exists $g : \mathbb{N}^k \to \mathbb{N}$ computable in the standard representation such that $f = \pi \circ g \circ \pi^{-1}$,*
2. *f is computable in (S, σ).*

Proof.

Let (S, σ) be generated by π.

We shall prove that (1) $\Rightarrow$ (2).

Let $f = \pi \circ g \circ \pi^{-1}$, where g is computable in the standard representation. We want to show that f is computable in (S, σ). Without loss of generality we can assume that $S = N$ and that (S, σ) is the canonical representation generated by π. Then for every $n \in \mathbb{N}$: $\sigma(\overline{n}) = \pi(n)$.

The function f can be computed by the following algorithm:

Read $\overline{n_1}, ..., \overline{n_k}$ — a sequence of numerals from $S = N$ as input, representing numbers $m_1, ..., m_k$ in (S, σ).

Return $\overline{g(n_1, ..., n_k)}$ as output.

This is computable because we assumed that g is computable in the standard representation. We need to show that this is an algorithm for f in (S, σ).

The numerals given as input: $\overline{n_1}, ..., \overline{n_k}$ represent respectively numbers $\sigma(n_1), ..., \sigma(n_k)$, i.e. $\pi(n_1), ..., \pi(n_k)$. The result returned as output represents the number:

$$\sigma(\overline{g(n_1, ..., n_k)}) = \pi(g(n_1, ..., n_k)) = \pi(g(\pi^{-1}(m_1), ..., \pi^{-1}(m_k))).$$

Hence this is an algorithm for f in (S, σ).

We shall prove that (2) $\Rightarrow$ (1).

Suppose that f is computable in (S, σ). Consider an algorithm which computes it. Let us observe that this algorithm can also be interpreted as computing a certain function g in the standard representation of $\mathbb{N}$. By using the same argument as in the proof of the implication in opposite direction, we can show that $f = \pi \circ g \circ \pi^{-1}$.

■

Theorem 5.12 *If (S, σ) is a representation of $\mathbb{N}$ with computable $\chi_=$ generated by π, then for any $R \subseteq \mathbb{N}^k$ the following conditions are equivalent:*

1. *there exists $T \subseteq \mathbb{N}^k$ computable in the standard representation such that for all $n_1, , , , , n_k$:*

$$R(n_1, ..., n_k) \Leftrightarrow T(\pi^{-1}(n_1), ..., \pi^{-1}(n_k)),$$

2. *R is computable in (S, σ).*

Proof.

Due to Corollary 5.4 and Theorem 3.11 we can assume without loss of generality that (S, σ) is the canonical representation generated by π. Then for every natural number n, $\sigma(\overline{n}) = \pi(n)$.

For any $R \subseteq \mathbb{N}^k$ there exists $T \subseteq \mathbb{N}^k$ such that:

$$R(n_1, ..., n_k) \Leftrightarrow T(\pi^{-1}(n_1), ..., \pi^{-1}(n_k)).$$

To prove this theorem it suffices to show that R is computable in (S, σ) if and only if T is computable in the standard representation.

For any standard numerals $\overline{n_1}, ..., \overline{n_k}$ the following conditions are equivalent:

1. $(\overline{n_1}, ..., \overline{n_k}) \in R^{\sigma}$,
2. $(\pi(n_1), ..., \pi(n_k)) \in R$,
3. $(\pi^{-1}(\pi(n_1)), ..., \pi^{-1}(\pi(n_k))) \in T$,
4. $(n_1, ..., n_k) \in T$,
5. $(\overline{n_1}, ..., \overline{n_k}) \in T^{\sigma_{st}}$.

This means that the same algorithm which computes R in (S, σ) also computes T in the standard representation.

■

We would also like to formulate and prove a more general version of Theorem 4.14.

Definition 5.13 *Let (N, σ_π) be the canonical representation of $\mathbb{N}$ generated by π. Then the following function:*

$$\pi Succ = \pi \circ Succ \circ \pi^{-1}$$

shall be called π-successor.

Corollary 5.14 *For every canonical representation (N, ν) of $\mathbb{N}$, generated by π, and every natural number n:*

$$\overline{n+1} =^{\nu} \pi Succ^{\nu}(\overline{n}).$$

Theorem 5.15 *In every representation (S, σ) of $\mathbb{N}$ generated by π, the π-successor function is computable.*

Proof.

If (S, σ) is generated by π, then it is isomorphic to the canonical representation (N, ν_π) and as a result in both these representations exactly the same functions are computable. Due to Theorem 5.11 the π-successor function is computable in (N, ν_π), so it must also be computable in (S, σ).

■

Theorem 5.16 *Let (S, σ) and (T, τ) be representations of $\mathbb{N}$, generated respectively by π_1 and π_2. The following conditions are equivalent:*

1. $(S, \sigma) \cong (T, \tau)$,
2. $\pi_1 Succ$ *is computable in* (T, τ),
3. $\pi_2 Succ$ *is computable in* (S, σ).

Proof.

Implications (1) ⇒ (2) and (1) ⇒ (3) are easy conclusions from Theorem 5.15. We shall show that (2) ⇒ (1) holds. The proof of (3) ⇒ (1) is analogous.

Suppose that $\pi_1 Succ$ is computable in (T, τ). Due to Corollary 5.4 we can assume without loss of generality that both (S, σ) and (T, τ) are canonical representations generated respectively by π_1 and π_2. Hence $\sigma(\overline{n}) = \pi_1(n)$ and $\tau(\overline{n}) = \pi_2(n)$, for every natural number n.

Also, since $\chi_=$ is computable in both considered representations, it suffices to demonstrate the existence of a computable translation function in one direction (this is a consequence of Theorem 6.3).

We shall show how to construct the translation function F^σ_τ.

Since (S, σ) is canonical, the number $\pi_1(0)$ is represented in it by $\overline{0}$. Let α_0 be the numeral of (T, τ) representing $\pi_1(0)$. Then

$$F^\sigma_\tau(\overline{0}) = \alpha_0$$

and for every natural number n:

$$F^\sigma_\tau(\overline{n+1}) = \pi_1 Succ(F^\sigma_\tau(\overline{n})),$$

since

$$\overline{n+1} =^{\sigma} \pi_1 Succ^{\sigma}(\overline{n}).$$

It follows that (S, σ) and (T, τ) are isomorphic.

■

The π-successor function serves as an equivalent of successor which can be computed in every representation generated by π. Similarly, for every function computable in the standard representation, we can define its equivalent for any representation with computable $\chi_=$. For any function $f : \mathbb{N}^n \to \mathbb{N}$, let:

$$\pi f = \pi \circ f \circ \pi^{-1}.$$

Correlations between computability of functions are exactly the same as those between computability of their π-equivalents. For example, if π-addition is computable in a given representation, then π-successor also is.[1] This shows that all representations with computable $\chi_=$ have very similar structure.

5.2 Computability of permutations

In this section we are going to consider computability of permutations in representations of $\mathbb{N}$. We want to show whether computability of a permutation π in a given representation implies computability of the reverse permutation π^{-1} in that representation. It turns out that the answer depends on whether we assume computability of $\chi_=$ or not. This hardly comes as a surprise, since it bears similarity to several other results established earlier, especially in section 4.2.

Theorem 5.17 *Let (S, σ) be a representation of $\mathbb{N}$ and π be a permutation of $\mathbb{N}$. If both π and $\chi_=$ are computable in (S, σ), then π^{-1} is also computable in (S, σ).*

Proof.

We shall use Theorem 5.11 in this proof. If π is computable in (S, σ) then there is a function g computable in the standard representation and a permutation ρ of $\mathbb{N}$ such that:

$$\pi = \rho \circ g \circ \rho^{-1}.$$

[1] This can be treated as an analogue of Theorem 4.16.

Then

$$\pi^{-1} = (\rho \circ g \circ \rho^{-1})^{-1} = \rho \circ g^{-1} \circ \rho^{-1}.$$

Note that g^{-1} exists because g is a permutation. We conclude that π^{-1} is computable in (S, σ).

■

Definition 5.18 *A sequence of numbers $(A_n)_{n\in\mathbb{N}}$ dominates a sequence $(a_n)_{n\in\mathbb{N}}$ if $A_n \geq a_n$ for every $n \in \mathbb{N}$.*

Note that this definition diverges slightly from the one that can be found in some sources, such as [57] or [58].

Lemma 5.19 *There exists a sequence of positive natural numbers $(a_n)_{n\in\mathbb{N}}$ which is not dominated by any computable sequence of positive natural numbers.*

Proof.
The proof is straightforward via diagonalisation.

■

Lemma 5.20 *Let $(a_n)_{n\in\mathbb{N}}$ be a sequence of positive natural numbers. Let $(b_n)_{n\in\mathbb{N}}$ be defined as follows:*

$$(b_n)_{n\in\mathbb{N}} = (\underbrace{a_0, ..., a_0}_{a_0 \text{ times}}, \underbrace{a_1, ..., a_1}_{a_1 \text{ times}}, ...).$$

If there exists $(B_n)_{n\in\mathbb{N}}$ — a computable sequence which dominates $(b_n)_{n\in\mathbb{N}}$, then there exists $(A_n)_{n\in\mathbb{N}}$ — a computable sequence which dominates $(a_n)_{n\in\mathbb{N}}$.

Proof.
Suppose that there is $(B_n)_{n\in\mathbb{N}}$ — a computable sequence which dominates $(b_n)_{n\in\mathbb{N}}$. Let us observe that $b_0 = a_0$ and let:

$$A_0 := B_0 \geq b_0 = a_0.$$

Let us observe that since $a_0 \leq A_0$, it follows that among the first $A_0 + 1$ elements of $(B_n)_{n\in\mathbb{N}}$, there must be a B_j such that $B_j \geq a_1$. Therefore let:

$$A_1 := \sup\{B_i : 0 \leq i \leq A_0\} \geq a_1.$$

Similarly, among the first $A_0 + A_1 + 1$ elements of $(B_n)_{n\in\mathbb{N}}$, there must be B_{j_1} and B_{j_2} such that $B_{j_1} \geq a_1$ and $B_{j_2} \geq a_2$. Let:

$$A_2 := \sup\{B_i : 0 \leq i \leq A_0 + A_1\} \geq a_2.$$

By extending this argument, for any positive natural number n let:

$$A_n := \sup\{B_i : 0 \leq i \leq \sum_{j=0}^{n-1} A_j\} \geq a_n.$$

■

Theorem 5.21 *There exist a representation (S, σ) and a permutation π of $\mathbb{N}$ computable in (S, σ) such that the permutation π^{-1} is not computable in (S, σ).*

Proof.

The notations and terminology used throughout this proof are the same as in the above lemmas. Let the alphabet Σ consist of all the standard digits $\overline{0}, ..., \overline{9}$, an additional symbol $\overline{\pi}$ and brackets (,). All standard numerals are members of S. Additionally, if $\alpha \in S$, then $\overline{\pi}(\alpha) \in S$.

Let $(a_n)_{n\in\mathbb{N}}$ be a sequence of positive natural numbers which is not dominated by any computable sequence. The permutation π shall consist of the following cycles:

$$(0, 1, ..., a_0 - 1),$$

$$(a_0, a_0 + 1, ..., a_0 + a_1 - 1),$$

$$(a_0 + a_1, a_0 + a_1 + 1, ..., a_0 + a_1 + a_2 - 1),$$

$$\vdots$$

$$(a_0 + a_1 + ... + a_n, a_0 + a_1 + ... + a_n + 1, ..., a_0 + a_1 + ... + a_n + a_{n+1} - 1),$$

$$\vdots$$

Thus, consecutive elements of the sequence $(a_n)_{n\in\mathbb{N}}$ are lengths of consecutive cycles of π.

The function σ interprets all the standard numerals in the usual manner. Furthermore, for any numeral α: $\sigma(\overline{\pi}(\alpha)) = \pi(\sigma(\alpha))$.

We shall show that if the permutation π^{-1} is computable, than we can construct a computable sequence $(B_n)_{n\in\mathbb{N}}$ defined as in the lemma above.

Let us observe that for every natural number n, b_n is the length of the cycle of π to which n belongs.

We adopt the convention that $\pi^k(n)$ and $\pi^{-k}(n)$ shall denote, respectively permutations π and π^{-1} iterated k times. Moreover, $\overline{\pi}^k(\overline{n})$ shall be an abbreviation for the following numeral:

$$\underbrace{\overline{\pi}(...(\overline{\pi}}_{k \text{ times}}(\overline{n})...)).$$

For the sake of convenience we shall also adopt a convention that $\pi^k(n) = n$ and $\overline{\pi}^k(\overline{n}) = \overline{n}$ for$k = 0$.

The following algorithm computes B_n for a given natural number n. Let us calculate $(\pi^{-1})^\sigma(\overline{n}), (\pi^{-2})^\sigma(\overline{n})), ...$ until any of the following occurs:

1. At a certain step of the algorithm, we get a result of the form

   $$(\pi^{-k})^\sigma(\overline{n}) = \overline{\pi}^l(\overline{n}).$$

   Then:

   $$n = \pi^{k+l}(n)$$

   and $b_n \leq k + l$. Therefore let:

   $$B_n := k + l.$$

2. At a certain step of the algorithm, we get a result of the form

   $$(\pi^{-k})^\sigma(\overline{n}) = \overline{\pi}^l(\overline{m}),$$

   for $m > n$. Then:

   $$\pi^{-k}(n) = \pi^l(\pi^{m-n}(n))$$

   and

   $$n = \pi^{k+l+m-n}(n).$$

   Since $b_n \leq k + l + m - n$, let

   $$B_n := k + l + m - n.$$

3. Suppose that none of the above situations occurs at any step of the algorithm. Then at every step we get a result of the form

$$(\pi^{-k})^{\sigma}(\overline{n}) = \overline{\pi}^{l}(\overline{m}),$$

for $m < n$. Let N be the number of all distinct numbers $\overline{m}$ such that $m < n$ and at a certain step of the algorithm we get the $\overline{\pi}^{l}(\overline{m})$ as the output, for any $l \in \mathbb{N}$. Let $m_1, ..., m_N$ be an enumeration of all these numbers.

For each $i = 1, ..., N$, by k_i we shall denote the first step at which $\overline{\pi}^{l}(\overline{m_i})$ is returned as the output, for any $l \in \mathbb{N}$. For each $i = 1, ..., N$, let:

$$(\pi^{-k_i})^{\sigma}(\overline{n}) = \overline{\pi}^{l_i}(\overline{m_i})$$

and hence:

$$n = \pi^{k_i + l_i}(m_i).$$

We are looking for such k_i', l_i' that

$$(\pi^{-k_i'})^{\sigma}(\overline{n}) = \overline{\pi}^{l_i'}(\overline{m_i}),$$

for a certain m_i, and $k_i' + l_i' > k_i + l_i$.

Let:

$$K = \max\{k_i + l_i : i = 1, ..., N\} + 1.$$

Note that we will find k_i', l_i' as described above at the latest after performing K steps of the algorithm. Then:

$$n = \pi^{k_i + l_i}(m_i) = \pi^{k_i' + l_i'}(m_i)$$

and, as a consequence:

$$m_i = \pi^{k_i' + l_i' - k_i - l_i}(m_i).$$

Hence, since $k_i' + l_i' - k_i - l_i > 0$, let:

$$B_n := k_i' + l_i' - k_i - l_i.$$

One of the above cases needs to occur at a certain point. Therefore, there exists a computable sequence of positive natural numbers $(B_n)_{n \in \mathbb{N}}$ which dominates $(b_n)_{n \in \mathbb{N}}$. Due to the lemma there also has to exist a sequence $(A_n)_{n \in \mathbb{N}}$ which dominates $(a_n)_{n \in \mathbb{N}}$. But this contradicts our assumption. Therefore, π^{-1} is not computable in (S, σ).

■

6

Criteria for comparing representations

In Chapter 3 we discussed criteria for similarity of representations. Here we wish to define criteria which can be used to compare representations in order to be able to decide which representations are in some way better or more powerful than the other ones.

6.1 Translatability

Translatability of representations is a one-sided equivalent of an isomorphism. A representation is translatable to another representation if there exists a computable translation from the former to the latter.

Definition 6.1 *Let (S, σ) and (T, τ) be representations of Z. Then (S, σ) is translatable to (T, τ) of Z if there exists a computable function (translation) F_τ^σ such that:*

$$\forall_{\alpha \in S} \exists_{\beta \in T} \ (F_\tau^\sigma(\alpha) = \beta \wedge \sigma(\alpha) = \tau(\beta)).$$

Corollary 6.2 *Representations (S, σ) and (T, τ) are isomorphic if and only if both (S, σ) is translatable to (T, τ) and (T, τ) is translatable to (S, σ).*

Theorem 6.3 *If (S, σ) is translatable to (T, τ) and $\chi_=$ is computable in (T, τ), then:*

1. *$\chi_=$ is computable in (S, σ),*

2. *(T, τ) is translatable to (S, σ).*

3. *(S, σ) and (T, τ) are isomorphic.*

Proof.

First we want to prove that $\chi_=$ is computable in (S, σ). Let $\alpha_1, \alpha_2 \in S$. The following conditions are equivalent:

1. $\chi_=(\alpha_1, \alpha_2) = TRUE$,

2. $\alpha_1 =^\sigma \alpha_2$,

3. $\tau(F^\sigma_\tau(\alpha_1)) = \tau(F^\sigma_\tau(\alpha_2))$,

4. $\chi_=(F^\sigma_\tau(\alpha_1), F^\sigma_\tau(\alpha_2)) = TRUE$.

It follows that $\chi_=$ is computable in (S, σ).

Now we want to construct an algorithm computing F^τ_σ — a translation from (T, τ) to (S, σ). Let $\alpha \in T$. Let us check $\zeta_0, \zeta_1, ...$ — numerals from S, one by one, and for each of them calculate $\beta_i = F^\sigma_\tau$ and then $\chi_=(\alpha, \beta_i)$ until we find such ζ_i that the value of the latter function is $TRUE$. This has to occur at a certain point because (S, σ) and (T, τ) are both representations of the same set.

Then let $F^\tau_\sigma(\alpha) = \zeta_i$.

Hence (T, τ) is translatable to (S, σ) and — as an immediate conclusion from Corollary 6.2 — these representations are isomorphic.

■

Computability of $\chi_=$ in (T, τ) is an essential assumption in this theorem. Even if we replaced this assumption with an analogous one for (S, σ), the theorem would no longer hold.

Theorem 6.4 *There exist (S, σ) and (T, τ) — representations of $\mathbb{N}$ such that:*

1. *$\chi_=$ is computable in (S, σ) but not in (T, τ),*

2. *(S, σ) is translatable to (T, τ) but not the other way round.*

Proof.

Let (S, σ) be the standard representation and let $A \subseteq \mathbb{N}$ be a set not computable in the standard representation. (T, τ) shall be defined as follows:

$$T = \{(\overline{a}, \overline{b}) : a \in \mathbb{N} \wedge b \in \{0, 1\}\}$$

and for every $a \in \mathbb{N}$:

$$\tau((\overline{a}, \overline{0})) = a,$$

$$\tau((\overline{a}, \overline{1})) = \begin{cases} 1 & \text{if } a \in A, \\ 0 & \text{if } a \notin A. \end{cases}$$

Obviously, $\chi_=$ is computable in (S, σ). It is not computable in (T, τ). Suppose to the contrary that it is. Let $a \in A$. Then for every natural number n the following conditions are equivalent:

1. $n \in A$,
2. $\tau((\overline{n}, \overline{1})) = 1$,
3. $(\overline{n}, \overline{1}) =^\tau (\overline{a}, \overline{1})$,
4. $\chi_=(\tau((\overline{n}, \overline{1})), \tau((\overline{a}, \overline{1}))) = TRUE$.

It follows that A is computable in the standard representation. Since this contradicts our assumption, we conclude that $\chi_=$ is not computable in (S, σ).

Representation (S, σ) is translatable to (T, τ), since the translation can be defined as follows:

$$F^\sigma_\tau(\overline{n}) = (\overline{n}, \overline{0}).$$

Suppose that (T, τ) is translatable to (S, σ) as well and let F^τ_σ be a translation. We shall show that this assumption leads to a contradiction.

For every natural number n, the following conditions are equivalent:

1. $n \in A$,
2. $\tau((\overline{n}, \overline{1})) = 1$,
3. $\tau((\overline{n}, \overline{1})) = \sigma(\overline{1})$
4. $F^\tau_\sigma((\overline{n}, \overline{1}) = \overline{1}$.

The last transition in the above sequence of equivalences is valid since $\overline{1}$ is the only numeral representing number 1 in (S, σ).

We conclude that A is computable in the standard representation. This, however, contradicts our assumption and hence (T, τ) is not translatable to (S, σ).

■

6.2 Reducibility and χ-reducibility

Definition 6.5 *Let (S, σ) and (T, τ) be representations of Z. We shall say that (S, σ) is reducible to (T, τ) if for every function f computable in (S, σ), it is also computable in (T, τ).*

Definition 6.6 *Let (S, σ) and (T, τ) be representations of Z. We shall say that (S, σ) is χ-reducible to (T, τ) if for every $R \subseteq Z^k$, if χ_R is computable in (S, σ), it is also computable in (T, τ).*

The concepts of reducibility and χ-reducibility of representations defined above are analogues of equivalence and χ-equivalence in the following sense.

Corollary 6.7 *Let (S, σ) and (T, τ) be representations of Z. Then:*

1. *(S, σ) and (T, τ) are equivalent if and only if (S, σ) is reducible to (T, τ) and (T, τ) is reducible to (S, σ),*
2. *(S, σ) and (T, τ) are χ-equivalent if and only if (S, σ) is χ-reducible to (T, τ) and (T, τ) is χ-reducible to (S, σ).*

Reducibility and χ-reducibility of representations are not equivalent to each other.

Theorem 6.8 *There exist representations (S, σ) and (T, τ) of $\mathbb{N}$ such that (S, σ) is reducible to (T, τ), but is not χ-reducible.*

Proof.

Let (S, σ) be the standard representation of $\mathbb{N}$, let F be the (countable) family of all functions computable in the standard representation and let $A \subseteq \mathbb{N}$, $A \neq \emptyset$, $A \neq \mathbb{N}$ be a set such that χ_A is computable in the standard representation. According to Theorem 4.26 there exists a representation

(T, τ) in which all functions from F are computable but χ_A is not. Then (S, σ) is reducible to such (T, τ), but is not χ-reducible.

■

We want to examine relation between translatability and new notions introduced in this section.

Theorem 6.9 *Let (S, σ) and (T, τ) be representations of Z such that $\chi_=$ is computable in (T, τ). If (S, σ) is translatable to (T, τ), then these representations are reducible and χ-reducible to each other.*

Proof.

Due to Theorem 6.3 it follows from these assumptions that both representations are isomorphic. Then they are also equivalent and χ-equivalent, due to Theorem 3.14. It follows from 6.7 that they are reducible and χ-reducible to each other.

■

The assumption of computability of $\chi_=$ in (T, τ) is crucial here. Without it, we can only prove the following implication:

Theorem 6.10 *If (S, σ) and (T, τ) are representations of Z and (S, σ) is translatable to (T, τ), then (T, τ) is χ-reducible to (S, σ).*

Proof.

Suppose $R \subseteq \mathbb{N}^n$ is computable in (T, τ). Let $\alpha_1, ..., \alpha_n \in S$. We want to know whether $(\alpha_1, ..., \alpha_n) \in R^\sigma$. It suffices that we check whether $(F^\sigma_\tau(\alpha_1), ..., F^\sigma_\tau(\alpha_n)) \in R^\tau$.

■

The other implications do not hold even if we additionally assume computability of $\chi_=$ in (S, σ).

Theorem 6.11 *There exist representations (S, σ) and (T, τ) of $\mathbb{N}$ such that $\chi_=$ is computable in (S, σ) and (S, σ) is translatable to (T, τ), but:*

1. *(S, σ) is not χ-reducible to (T, τ),*
2. *neither representation is reducible to the other.*

Proof.

Consider representations defined in the proof of Theorem 6.4. We have already proved that $\chi_=$ is computable in (S, σ) and that (S, σ) is translatable to (T, τ). We want to show that these representations also satisfy all the other conditions.

Representation (S, σ) cannot be χ-reducible to (T, τ). Otherwise they would both be χ-equivalent (due to Corollary 6.7) and $\chi_=$ would be computable in (T, τ) — but we know that this is not the case.

We want to show that (T, τ) is not reducible to (S, σ). Let us define the following function f:

$$f(n) = \begin{cases} 0 & \text{if } n > 1 \wedge n \notin A, \\ 1 & \text{if } n > 1 \wedge n \in A, \\ 2 & \text{if } n = 0 \vee n = 1. \end{cases}$$

The function f is computable in (T, τ). Let:

$$f^\sigma((\overline{0}, \overline{0})) = f^\sigma((\overline{1}, \overline{0})) = (\overline{2}, \overline{0}),$$

$$f^\sigma((\overline{n}, \overline{0})) = (\overline{n}, \overline{1}), \text{ for every } n \geq 2.$$

$$f^\sigma((\overline{n}, \overline{1})) = (\overline{2}, \overline{0}), \text{ for every } n.$$

Function f^σ is a computable representation of f in (S, σ).

However, f is not computable in (S, σ) (i.e. the standard representation). If it were, then A would also need to be computable in the standard representation, contrary to the assumption. For any $n \neq 2$ we would need to calculate $f(n)$ and then check whether the value of the function is 0 or 1. The answers for n equal to 0 or 1 could be given explicitly as special cases.

We also want to show that (S, σ) is not reducible to (T, τ) either. The successor function $Succ$ is obviously computable in (S, σ). We shall show that it is not computable in (T, τ).

For any $n \in \mathbb{N}$, the following conditions are equivalent:

1. $n \in A$,
2. $\tau((\overline{n}, \overline{1})) = 1$,
3. $\tau(Succ^\tau((\overline{n}, \overline{1}))) = 2$,
4. $Succ^\tau((\overline{n}, \overline{1})) = (\overline{2}, \overline{0})$.

Hence, if successor was computable in (T, τ), then A would be computable in the standard representation. That would be a contradiction.

We conclude that neither representation is reducible to the other.

■

6.3 Stronger and weaker representations

Given two representations of the same set, we want to be able to decide if one of them is in some way 'better' than the other. A possible way of handling this is to show that a certain representation allows us to compute more functions or sets (in terms of inclusion).

Definition 6.12 *Let (S, σ) and (T, τ) be representations of $\mathbb{N}$. Then we shall say that:*

1. *(S, σ) is at least as strong as (T, τ) if (T, τ) is both reducible and χ-reducible to (S, σ),*
2. *(S, σ) is at least as weak as (T, τ) if (T, τ) is at least as strong as (S, σ),*
3. *(S, σ) is (strictly) stronger than (T, τ) if (S, σ) is at least as strong as (T, τ) and there exists a function (numerical or Boolean) computable in (S, σ) but not in (T, τ),*
4. *(S, σ) is (strictly) weaker than (T, τ) if (T, τ) is (strictly) stronger than (S, σ),*
5. *(S, σ) and (T, τ) are incomparable if neither (S, σ) is at least as strong as (T, τ) nor (T, τ) is at least as strong as (S, σ).*

Theorem 6.13 *If (S, σ) and (T, τ) are representations of $\mathbb{N}$ with computable $\chi_=$, then they are either isomorphic or incomparable.*

Proof.

Suppose that representations (S, σ) and (T, τ) with computable $\chi_=$ are comparable (i.e. not incomparable). We want to show that they are isomorphic.

It follows from our assumption that one of these representations is at least as strong as the other. We can assume without loss of generality that (T, τ)

is at least as strong as (S, σ). We know that every function computable in (S, σ) is also computable in (T, τ), in particular $\pi Succ = \pi \circ g \circ \pi^{-1}$, where π is a permutation generating (S, σ). Then, by Theorem 5.16 we conclude that (S, σ) and (T, τ) are isomorphic.

■

Theorem 6.14 *There exists no representation of* $\mathbb{N}$ *stronger than the standard representation.*

Proof.

Suppose to the contrary such a representation (S, σ) exists. By previous theorem, it cannot have computable $\chi_=$. However, $\chi_=$ is computable in the standard representation but not in (S, σ), thus the standard representation is not χ-reducible to (S, σ). This is a contradiction. Therefore no representation of $\mathbb{N}$ can be stronger than the standard representation.

■

The above theorem can be interpreted in such a way that the standard representation of $\mathbb{N}$ cannot be improved. The same theorem and proof are also valid for every other representation of natural numbers with computable $\chi_=$.

7

Turing reducibility of sets and functions in representations

The idea behind Turing reducibility is following: how to solve a certain problem A if we already possess some (possibly non-algorithmic) means to know the solution of problem B. This is a very important concept in computability theory. More information about this subject can be found in [50] and [58].

In this chapter we are going to consider the notion of Turing reducibility relativised to a representation.

7.1 Turing reducibility of sets

All problems considered in this section will be of the following form: for a given $R \subseteq \mathbb{N}^n$ (where n is a positive natural number) decide for any $a_1, ..., a_n \in \mathbb{N}$, whether $(a_1, ..., a_n) \in R$ or not. To solve a problem means to construct an algorithm returning the correct answer $TRUE$ or $FALSE$ for any sequence of numbers given as input.

While this is the standard approach to Turing reducibility of sets, we are going to relativise all the notions to a chosen representation.

For the sake of simplicity, we limit our considerations in this section to

subsets of $\mathbb{N}$.

Definition 7.1 *Let (S, σ) be a representation of $\mathbb{N}$ and let $A, B \subseteq \mathbb{N}$. We say that A is Turing reducible to B (or B-computable, computable with an oracle B) in (S, σ) if there exists an algorithm which:*

1. *reads a numeral $\alpha \in S$,*
2. *returns $TRUE$ if and only if $\sigma(\alpha) \in A$, otherwise it returns $FALSE$,*
3. *in addition to the usual instructions, it has an instruction which, for any variable v and any numeral $\beta \in S$, assigns $TRUE$ to v if and only if $\sigma(\beta) \in B$, otherwise it assigns $FALSE$ to v.*

We write: $A \leq_T^\sigma B$. If (S, σ) is the standard representation, we can simply write $A \leq_T B$.

Definition 7.2 *Let (S, σ) be a representation of $\mathbb{N}$ and $A, B \subseteq \mathbb{N}$. If both $A \leq_T^\sigma B$ and $B \leq_T^\sigma A$, then we say that A and B are Turing equivalent in (S, σ) and we write: $A \equiv_T^\sigma B$.*

Corollary 7.3 *For every representation (S, σ), $\equiv_T^\sigma$ is an equivalence relation on $\mathbb{N}$.*

Definition 7.4 *For every representation (S, σ) of $\mathbb{N}$, the equivalence classes of $\equiv_T^\sigma$ are called σ-Turing degrees. The Turing degree of a set A is written $\sigma deg(A)$. Furthermore:*

$$\sigma DEG = \{\sigma deg(A) : A \subseteq \mathbb{N}\}.$$

If (S, σ) is the standard representation, we can simply write $deg(A)$ instead of $\sigma deg(A)$ and DEG instead of σDEG.

Definition 7.5 *If $A, B \subseteq \mathbb{N}$ and (S, σ) is a representation of $\mathbb{N}$, then we define:*

$$deg(A) \leq_T^\sigma deg(B) \Leftrightarrow A \leq_T^\sigma B.$$

Theorem 7.6 *Let (S, σ) and (T, τ) be isomorphic representations of $\mathbb{N}$ with computable $\chi_=$ and let $A, B \subseteq \mathbb{N}$. Then:*

$$A \leq_T^\sigma B \Leftrightarrow A \leq_T^\tau B.$$

Proof.

We shall show the implication in one direction.

Suppose that $A \leq_T^\sigma B$. There exists an algorithm e computing A with an oracle B in (S, σ). For every $\alpha \in T$ we want to know whether $\tau(\alpha) \in A$ and we can utilise a τ-oracle for B.

To this aim:

1. We calculate $\beta = F_\sigma^\tau(\alpha)$.
2. We check whether $\tau(\alpha) \in A$ by performing the algorithm e, with β as the input, modified in the following way: we replace every instruction asking the oracle whether $\sigma(\lambda) \in B$ with an instruction asking the oracle whether $\tau(F_\tau^\sigma(\lambda)) \in B$.

■

Theorem 7.7 *Let (S, σ) be a representation of $\mathbb{N}$ generated by π and let $A, B \subseteq \mathbb{N}$. Then:*

$$A \leq_T^\sigma B \Leftrightarrow \pi^{-1}(A) \leq_T \pi^{-1}(B).$$

Proof.

Without loss of generality we can assume that (S, σ) is the canonical representation generated by π. We can do so due to Corollary 5.4 and because isomorphism preserves computability with an oracle (Theorem 7.6).

($\Rightarrow$) Suppose $A \leq_T^\sigma B$. Then there exists an algorithm computing A with an oracle B in (S, σ).

Note that the algorithm e also computes $\pi^{-1}(A)$ with an oracle $\pi^{-1}(B)$ in the standard representation. In both cases the oracle asks the same question, just phrased in a different way, because:

$$\sigma(\overline{n}) \in B \Leftrightarrow \pi(n) \in B \Leftrightarrow n \in \pi^{-1}(B),$$

by definition of the canonical representation.

Also, e returns the correct answer under this interpretation because for any natural number n the following conditions are equivalent:

1. $\chi_A^\sigma(\overline{n}) = TRUE$,
2. $\sigma(\overline{n}) \in A$,

3. $\pi(n) \in A$,

4. $n \in \pi^{-1}(A)$,

5. $\chi_{\pi^{-1}(A)}(n) = TRUE$.

($\Leftarrow$) We use a similar argument. Suppose $\pi^{-1}(A) \leq_T \pi^{-1}(B)$. Then there exists an algorithm computing $\pi^{-1}(A)$ with an oracle $\pi^{-1}(B)$ in the standard representation. The same algorithm also computes A with an oracle B in (S, σ), only questions asked by the oracle need to be reworded like above. To reach this conclusion, we consider the same sequence of equivalent conditions as in the proof of the other implication.

■

These are easy conclusions from the above theorem:

Corollary 7.8 *Let (S, σ) be a representation of $\mathbb{N}$ generated by π and let $A, B \subseteq \mathbb{N}$. Then:*

$$A \equiv_T^\sigma B \Leftrightarrow \pi^{-1}(A) \equiv_T \pi^{-1}(B).$$

Corollary 7.9 *Let (S, σ) be a representation of $\mathbb{N}$ generated by π and let $A \subseteq \mathbb{N}$. Then:*

$$\pi^{-1}(\sigma deg(A)) = deg(\pi^{-1}(A)).$$

Corollary 7.10 *For any representation (S, σ) of $\mathbb{N}$ with computable $\chi_=$:*

$$\sigma DEG = DEG.$$

Proof.

Considering that π^{-1} is a bijection on $\mathbb{N}$, and that the function assigning to each set its image under π^{-1} is a bijection on $\mathcal{P}(\mathbb{N})$, this is an easy conclusion from Corollary 7.9.

■

Theorem 7.11 *Let (S, σ) and (T, τ) be representations of $\mathbb{N}$ with computable $\chi_=$. Then there exist isomorphisms:*

1. $f : (\mathcal{P}(\mathbb{N}), \leq_T^\sigma) \to (\mathcal{P}(\mathbb{N}), \leq_T^\tau)$,

2. $g : (DEG, \leq_T^\sigma) \to (DEG, \leq_T^\tau)$.

In the above theorem, isomorphism is understood in the usual algebraic sense, as a bijective function between these structures, preserving their ordering.

Proof.

It suffices to show that such isomorphisms exist if (S, σ) is generated by π and (T, τ) is standard.

Let $f(A) = \pi^{-1}(A)$. This function is bijective since π^{-1} is a permutation of $\mathbb{N}$. Also, if $A \equiv_T^\sigma B$, then $\pi^{-1}(A) \equiv_T \pi^{-1}(B)$, according to Theorem 7.7, hence this is an isomorphism.

Isomorphism g shall be defined as follows:

$$\forall_{A \subseteq \mathbb{N}}\ g(\sigma deg(A)) = deg(\pi^{-1}(A)).$$

This is a well-defined function due to Corollary 7.8 and it is bijective. It is indeed an isomorphism since the following conditions are equivalent:

1. $\sigma deg(A) \leq_T^\sigma \sigma deg(B)$,
2. $A \leq_T^\sigma B$,
3. $\pi^{-1}(A) \leq_T \pi^{-1}(B)$,
4. $deg(\pi^{-1}(A)) \leq_T \deg(\pi^{-1}(B))$,
5. $g(A) \leq_T g(B)$.

■

The conclusion from this section is that the structure of Turing degrees in all representations with computable $\chi_=$ is the same. Therefore theorems proved for standard degrees also hold for degrees relativised to such a representation.

7.2 Turing reducibility of functions

In this section we are going to consider Turing reducibility of numerical functions. We wish to show that the notion of Turing reducibility of functions is much more problematic than in case of sets. The reason for this is that there can exist many answers to a question regarding a value of a certain numerical function if a representation is ambiguous. What is more, it turns

out that the choices of answers to such questions determine whether certain functions are Turing reducible to other functions or not.

The notion of Turing reducibility of functions in ambiguous representations can be defined in a number of non-equivalent ways. We intend to demonstrate unexpected and unintuitive properties of some of these definitions.

We assume that all functions under consideration in this section are unary.

By mimicking the definition of Turing reducibility of sets, we can define the reducibility of functions in the following way:

Definition 7.12 (attempt) *Let (S, σ) be a representation of $\mathbb{N}$ and let $f, g : \mathbb{N} \to \mathbb{N}$. We say that f is reducible to g (or computable with a g-oracle) in (S, σ) if there exists an algorithm which:*

1. *reads a numeral $\alpha \in S$,*
2. *returns a numeral $\alpha' \in S$ such that $\sigma(\alpha') = f(\sigma(\alpha))$,*
3. *in addition to the usual instructions, it has an instruction, which, for any variable v and any numeral $\beta \in S$, assigns a numeral $\beta' \in S$ such that $\sigma(\beta') = g(\sigma(\beta))$.*

We write: $f \leq_T^\sigma g$. If (S, σ) is the standard representation, we can simply write $f \leq_T g$.

Note that we have chosen to diverge slightly from the usual terminology and to use the term 'g-oracle' rather than 'oracle g'. The reason for this is that in ambiguous representations we are going to consider numerous oracles of the same function. By modifying standard terminology, we will be able to distinguish between these various oracles. More details will be given later in this section.

While the above definition is correct for unambiguous representations, a problem arises when there is a number represented by more than one numeral: who chooses the answer given by the algorithm whenever the additional instruction is utilised? And how is this answer chosen? We shall consider several possibilities.

Definition 7.13 *Let (S, σ) be a representation of $\mathbb{N}$ and let $f, g : \mathbb{N} \to \mathbb{N}$. We say that f can be computed by an algorithm with a g-oracle if for every input $\alpha \in S$ there exists a run of this algorithm which*

returns $\beta \in S$ such that $f(\sigma(\alpha)) = \sigma(\beta)$. Furthermore, we say that it can be computed:

1. *with a non-deterministic g-oracle (or by a non-deterministic algorithm with a g-oracle) if the oracle can change answer given to a question at any point, i.e. it can give different answers if asked the same question several times,*

2. *with a semi-deterministic g-oracle (or by a semi-deterministic algorithm with a g-oracle) if the oracle cannot change answer given to a question during a run of an algorithm, but can change them if the same algorithm is performed again (possibly with a different input), i.e. it can give different answers to the same question but only during different runs of the algorithm,*

3. *with a deterministic g-oracle (or by a deterministic algorithm with a g-oracle) if the oracle must always give the same answer to the same question during every run of the algorithm.*

When we speak of a 'run of an algorithm', we are referring to everything the algorithm does when it is executed. Two runs of a certain non-deterministic algorithm can be different if answers to the question given by the oracle differ.

It is our choice to say 'can compute' instead of 'computes' in the context of algorithms with non-deterministic or semi-deterministic oracles, since the answer given by the algorithm in at least some cases depends on the answers given by the oracle. Therefore, not every run of such an algorithm has to be a correct computation of the value of a function for the given input. Note that this distinction is not relevant for computations with deterministic oracles.

We can think of a g-oracle as a function $g^\sigma : S \to \mathcal{P}(S)$. Similarity to the notation for representations of functions is intended, since both notions refer to numeral-based equivalents of functions on natural numbers. Each g-oracle determines which answers to questions asked by an algorithm are possible. In this interpretation, the oracle can give answer β to a question about the value of $g^\sigma(\alpha)$ if $\beta \in g^\sigma(\alpha)$. In case of deterministic oracles, we can simply think of them as functions $g^\sigma : S \to S$.

It is also worth emphasising that being non-deterministic or semi-deterministic is not an inherent characteristic of the function g^σ itself. The same function $g^\sigma : S \to \mathcal{P}(S)$ can be used as either a non-deterministic

or a semi-deterministic oracle. The difference between these two types of oracles lies only in how an algorithm is allowed to make use of them (more specifically: when the algorithm is allowed to give different answers to the same question).

This interpretation of computability with an oracle provided here is obviously a very lenient one, and certainly not the only one possible.

Theorem 7.14 *Let (S, σ) be a representation of $\mathbb{N}$ and let $g : \mathbb{N} \to \mathbb{N}$ be such that for a certain natural number n, the number $g(n)$ is represented by at least two numerals in (S, σ). Then there exists a non-determinitic g-oracle g^σ such that every function $f : \mathbb{N} \to \mathbb{N}$ can be computed in (S, σ) with g^σ.*

Proof.
Let g^σ be such that for every $\alpha \in S$:

$$g^\sigma(\alpha) = \{\lambda \in S : \sigma(\lambda) = g(\sigma(\alpha))\}.$$

We want to calculate $f^\sigma(\alpha)$ with the oracle g^σ for $\alpha \in S$. Let $(\zeta_n)_{n\in\mathbb{N}}$ be a computable enumeration of S, and let β, β_1, β_2 be such that

$$g(\sigma(\beta)) = \sigma(\beta_1) = \sigma(\beta_2).$$

This algorithm can be used to compute f with the non-deterministic oracle g^σ:

$i \leftarrow 0$
$a \leftarrow g^\sigma(\beta)$
while $a = \beta_1$ **do**
    $i \leftarrow i + 1$ $a \leftarrow g^\sigma(\beta)$
return ζ_i

The above algorithm repeatedly asks about the value of $g^\sigma(\beta)$ and it counts how many times the answer β_1 was given (until the answer β_2 is given for the first time).

Suppose that $f^\sigma(\alpha) = \zeta_i$. Since the above algorithm is non-deterministic, it can give the answer β_1 to the oracle's question exactly i times and then give the answer β_2. Then the above algorithm will return the answer ζ_i as the output. Therefore, this algorithm can compute f with a non-deterministic g-oracle.

■

Theorem 7.15 *Let (S, σ) be a representation of $\mathbb{N}$ and let $g : \mathbb{N} \to \mathbb{N}$ be such that for a certain $n \in \mathbb{N}$, the number $g(n)$ is represented by a computably enumerable infinite set of numerals. Then there exists a semi-determinitic g-oracle g^σ such that every function $f : \mathbb{N} \to \mathbb{N}$ can be computed in (S, σ) with g^σ.*

Proof.
Let g^σ be such that for every $\alpha \in S$:

$$g^\sigma(\alpha) = \{\lambda \in S : \sigma(\lambda) = g(\sigma(\alpha))\}.$$

We wish to calculate $f^\sigma(\alpha)$ with the oracle g^σ, for a certain $\alpha \in S$. Let $(\zeta_k)_{k\in\mathbb{N}}$ be a computable enumeration of S. Let n be a number as in the theorem's assumptions, $\sigma(\beta) = n$, and let $S' \subseteq S$ be a subset of all numerals representing $g(n)$. There exists $(\beta_k)_{k\in\mathbb{N}}$ — a computable enumeration of S'. Let $\varphi : S' \to \mathbb{N}$ be such that $\varphi(\beta_k) = k$, for every β_k.

This algorithm can be used to compute f with the semi-deterministic oracle g^σ:

$a \leftarrow g^\sigma(\beta)$
$ind \leftarrow \varphi(a)$
return ζ_{ind}

Suppose that $f^\sigma(\alpha) = \zeta_i$. Then it suffices that the above algorithm gives the answer β_i to the oracle's question. Therefore, this algorithm can compute f with a semi-deterministic g-oracle.

■

Since from now on, we are going to be primarily concerned with deterministic oracles in this chapter, we wish to introduce some terminology regarding them. In Definition 2.6 we have established that every function on numbers g can be represented by a function on numerals g^σ. We have also concluded that, in case of ambiguous representations, many such functions g^σ exist. Now, we want to point out that each of them can be equated with a deterministic oracle.

If a representation g^σ of function g is invoked as an oracle by an algorithm, then we shall say that g^σ is an g-oracle (in a given representation). When we wish to distinguish between various g-oracles, we shall denote them as $g^\sigma_{(1)}, g^\sigma_{(2)}, ...$ — i.e. by adding indices in brackets.

If for every function f, it is computable with an oracle $g^\sigma_{(1)}$ iff it is computable with an oracle $g^\sigma_{(2)}$, then we shall call these oracles equivalent.

Theorem 7.16 *Let (S, σ) be a representation of $\mathbb{N}$ with computable $\chi_=$ and let $g : \mathbb{N} \to \mathbb{N}$ be such that:*

1. *there exists an infinite computably enumerable set $S' \subseteq S$ such that for every $\beta \in S'$, the number $g(\sigma(\beta))$ is represented by an infinite computably enumerable set of numerals in (S, σ),*

2. *there exists a computable function $\psi : S' \to S$ such that for every $\beta \in S'$: $\sigma(\psi(\beta)) = g(\sigma(\beta))$.*

Then for every function $f : \mathbb{N} \to \mathbb{N}$ there exists a deterministic g-oracle g^σ such that f is computable in (S, σ) with g^σ.

Proof.

We wish to calculate $f^\sigma(\alpha)$ with a deterministic g-oracle, for $\alpha \in S$. Let $(\zeta_k)_{k\in\mathbb{N}}$ be a computable enumeration of S, $(\beta_k)_{k\in\mathbb{N}}$ — a computable enumeration of S', and ψ — a function satisfying the second condition.

We want to construct an algorithm such as described in this theorem. To each possible input numeral we want to assign a different question that the oracle will ask. When supplied ζ_k, the oracle is going to ask about the value of $g^\sigma(\beta_k)$. An algorithm during each run will be able to ask the correct question, since both S and S' are computably enumerable.

Let:

$$G = \{(\lambda_1, \lambda_2) \in S^2 : \lambda_1 \in S' \wedge g(\sigma(\lambda_1)) = \sigma(\lambda_2)\}.$$

This set is computably enumerable because S' is computably enumerable and for each $\lambda_1 \in S'$ we know that $(\lambda_1, \psi(\lambda_1)) \in G$ and hence for each $(\lambda_1, \lambda_2) \in S^2$:

$$(\lambda_1, \lambda_2) \in G \Leftrightarrow \lambda_1 \in S' \wedge \lambda_2 =^\sigma \psi(\lambda_1).$$

Let us fix a certain computable enumeration of G. For each pair (λ_1, λ_2) in G, let $\varphi((\lambda_1, \lambda_2))$ be the number of pairs in G which have the same numeral λ_1 on the first position and come earlier in the given enumeration. Then φ is a computable function.

Consider the following algorithm with $\alpha = \zeta_k$ given as the input:

$a \leftarrow g^\sigma(\beta_k)$
$ind \leftarrow \varphi((\beta_k, a))$
return ζ_{ind}

Suppose that $f^\sigma(\alpha) = \zeta_i$. Let ζ_j be such that $\varphi((\beta_k, \zeta_j)) = i$. Such ζ_j exists for every i because due to assumptions there are infinitely many numerals representing number $g(\sigma(\beta_k))$. It suffices that the above algorithm gives answer ζ_j to the above question. Furthermore, for each input, a different question is asked, so there is no risk that for a different input, the oracle would have to give a different answer to the same question.

Hence this algorithm computes f with a deterministic g-oracle.

■

The assumptions in the above theorem might seem unintuitive, but they are easily satisfied.

Definition 7.17 *Let the representation (S, σ) of $\mathbb{N}$ be defined as follows:*

The alphabet Σ consists of standard digits $\overline{0}, ..., \overline{9}$, a left bracket (, a right bracket) and a comma. The set of numerals S consists of all inscriptions of the form $(\overline{a}, \overline{b})$, where $\overline{a}, \overline{b}$ are standard numerals.

For every $a, b \in \mathbb{N}$, let:

$$\sigma(\overline{a}, \overline{b}) = a.$$

The representation (S, σ), together with $S' = S$ and g, ψ — identity functions, satisfy all conditions of Theorem 7.16.

We can further generalise this result.

Theorem 7.18 *For every (S, σ) — representation of $\mathbb{N}$, there exists a representation (T, τ) such that:*

1. *$(S, \sigma) \cong (T, \tau)$,*
2. *for every function $f : \mathbb{N} \to \mathbb{N}$ there exists a deterministic g-oracle g^σ such that f is computable in (T, τ) with g^σ.*

Let $T = \{\alpha \frown X \frown \overline{n} : \alpha \in S \wedge n \in \mathbb{N}\}$, where X is any symbol not in S, and let $\tau(\alpha \frown X \frown \overline{n}) = \sigma(\alpha)$, for every $\alpha \in S$ and $n \in \mathbb{N}$. Obviously $(S, \sigma) \cong (T, \tau)$. Also (T, τ) satisfies all assumptions of Theorem 7.16, hence for every function $f : \mathbb{N} \to \mathbb{N}$ there exists a deterministic g-oracle g^σ such that f is computable in (T, τ) with g^σ.

Theorem 7.19 *There exist (S, σ) — a representation of $\mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$ such that certain deterministic g-oracles in (S, σ): $g^\sigma_{(1)}$, $g^\sigma_{(2)}$ are not equivalent.*

Proof.

Let (S, σ) and g be like in assumptions of Theorem 7.16. Then every $f : \mathbb{N} \to \mathbb{N}$ is computable in (S, σ) with a certain deterministic g-oracle. However, for every deterministic g-oracle g^σ, only countably many functions are computable with it. This is because there are only countably many algorithms and if the oracle is deterministic, then the run of each of these algorithms is also unambiguously determined.

Let f_1 be computable with $g^\sigma_{(1)}$ and f_2 — not computable with such oracle. Then f_2 is computable with another g-oracle, let us denote it as $g^\sigma_{(2)}$.

It follows that $g^\sigma_{(1)}$ and $g^\sigma_{(2)}$ are not equivalent.

■

Theorem 7.20 *There exist (S, σ), (T, τ) — isomorphic representations of $\mathbb{N}$ and functions $f, g : \mathbb{N} \to \mathbb{N}$ such that a certain non-deterministic (alternatively: semi-deterministic, or deterministic) algorithm can compute f with a g-oracle in (S, σ), but no non-deterministic (alternatively: semi-deterministic, or deterministic) can compute f with a g-oracle in (T, τ).*

Proof.

Let (S, σ) be the standard representation and (T, τ) be defined as in 7.17. Obviously these representations are isomorphic and they satisfy assumptions of, respectively, Theorem 7.14, 7.15 or 7.16. Let g be an identity function.

For every f: $f \leq^\sigma_T g$ if and only if f is computable in (S, σ). However, $f \leq^\tau_T g$, for every f, since according to the respective theorem every function can be computed in (T, τ) with a certain g-oracle.

It suffices to take any f uncomputable in (S, σ).

■

Theorem 7.21 *There exist (S, σ) — a representation of $\mathbb{N}$, functions $f, g : \mathbb{N} \to \mathbb{N}$ and $g^\sigma_{(1)}$, $g^\sigma_{(2)}$ — deterministic g-oracles such that f is computable in (S, σ) with oracle $g^\sigma_{(1)}$, but not with $g^\sigma_{(2)}$.*

Proof.

Let (S, σ) be like in Definition 7.17, f — any function uncomputable in (S, σ) and g — identity function.

According to Theorem 7.16, f is computable with a certain deterministic g-oracle $g^\sigma_{(1)}$. However, it is not computable with $g^\sigma_{(2)}$ defined as follows:

$$\forall_{\alpha \in S}\ g^\sigma_{(2)}(\alpha) = \alpha,$$

since then it would be computable without any oracle, contrary to the assumption. To compute f, we would need to take the algorithm which computes f with $g^{\sigma}_{(2)}$ and replace every instance of instruction of type:

$v \leftarrow g^{\sigma}_{(2)}(\alpha)$

with the instruction:

$v \leftarrow \alpha$.

■

Considerations contained in this section have shown that not all notions regarding computability can easily be generalised to all representations. The interpretation of Turing reducibility of functions developed here has unexpected computational properties. It allows us to prove that certain functions are reducible to certain other functions (in the sense defined here) even when their reducibility is based — to a large extent — not on intrinsic properties of functions or relations between them, but on tricks exploiting ambiguity of representations.

What is more, such reducibility is not preserved by an isomorphism of representations. In this context, Theorem 7.18 can be interpreted in the following way: we can largely extend the amount of functions computable with an oracle in a given representation by adding infinitely many new copies (indexed with natural numbers) of the same old numerals that have already been in that representation. This seems to be an undesirable and unintuitve property.

We do not claim that this is the only possible interpretation of Turing reducibility of functions in ambiguous representations. However, it is clear that this ambiguity is the source of many difficulties.

8

Philosophical perspective on representations

In this chapter we are going to consider questions whose nature is different than of those discussed earlier. While in preceding chapters we were inquiring into representations of numbers mainly from a mathematical perspective, here we wish to take a look at philosophical aspects of this subject. Even if these two perspectives are not always closely linked, we believe that they complement each other and hence both of them need to be taken into account. It is also worth emphasising that most authors dealing with this subject prefer to focus on its philosophical side — examples will be given later in this chapter.

8.1 General remarks on Church's thesis

8.1.1 Formulation and status of the thesis

The central place in philosophical discussions surrounding representations of natural numbers is occupied by Church's thesis. For this reason let us begin by formulating this thesis and considering its status. This section is largely

based on [30] and [23] (in Polish). More details on issues raised in this section can be found in these sources.

While different formulations of Church's thesis are possible, let us phrase it in the following way:

Thesis 8.1 (Church's thesis) *Intuitively computable functions are exactly those functions that are computable by Turing machines.*

The thesis can also be formulated in the following way:

Thesis 8.2 (Church's thesis) *Intuitively computable functions are exactly recursive functions.*

In either case, Church's thesis links a (quasi-)empirical notion of (intuitive) computability to a formal mathematical model of computations, such as Turing machines or recursivness. This can be utilised by mathematicians and philosophers in either of two different ways: inessential and essential.

Church's thesis is utilised in an inessential way when it is a way to speed up a proof which could be formulated without resorting to it. For example, when we wish to prove that a certain function is recursive, or computable by a Turing machine, it suffices to show that it is intuitively computable and then invoke the Church's thesis. This allows us to shorten proofs which would otherwise have been longer and more complicated.

However, sometimes if the proof would not have been possible without Church's thesis, this constitutes an essential use. Using Church's thesis in this way is especially common in philosophical investigations regarding mathematics. The examples are: determining the scope of usability of theorems regarding recursive functions (such as proving that Gödel's theorems are valid for all effective formal systems) or proving the undecidability of certain theories regardless of the chosen computability model. Later in this chapter, we will also make essential use of Church's thesis.

While Church's thesis — as its name could imply — should be considered a thesis, there is no consensus regarding its status. Various authors consider it to be:

1. a hypothesis or thesis,
2. a definition,
3. a model or explication,

4. an axiom,

5. a theorem.

More details on this debate can be found in the cited sources. In the following subsections we shall focus on the last of the suggested interpretations.

8.1.2 Arguments for Church's thesis

If Church's thesis is a theorem, then we should be able to prove it. However, while the general consensus seems to be that the thesis is true, it is not clear whether a proof of it exists. This is because its nature is not purely formal mathematical — it contains reference to a vague and not well-defined notion of intuitive computability.

It is argued in favour of Church's thesis that all major models of computability, such as Turing machines, recursive functions or λ calculus are equivalent. This suggests that they all capture some basic intuitions regarding the notion of computability — but perhaps it is just that their respective authors each made the same mistake or subconsciously all adopted the same assumption? While the equivalence of all these models does not prove Church's thesis, it speaks in favour of it.

Church's thesis consists of two implications. One of them ('every function computable by a Turing machine is intuitively computable') is usually considered rather uncontroversial, only the other one ('every intuitively computable function is computable by a Turing machine') stirs up the debate. Oddly enough, in context of representations the 'easy' part of Church's thesis is also problematic. Usual criticism regarding the 'easy' implication will be discussed in subsection 8.1.3. Arguments which arise specifically from introducing non-standard representations will be considered in section 8.3.

For Gödel the very construction of Turing machines appeared to be so convincing that he considered Church's thesis to be proved by it. Later, however, others tried to come up with various justifications.

The usual strategies employed in proving Church's thesis consisted in turning computability into a formal notion by identifying and formulating hidden assumptions about it. This was done among others by Andrey Kolmogorov [29], Robin Gandy [17] and Wilfried Sieg, among others in [55] and [56].

We are going to summarise Sieg's analysis here. This is based on [30].

At first, Sieg wanted to formulate a thesis which he called the Turing's Central Thesis and which was in his opinion sufficient to prove Church's thesis. According to this central thesis, every computation can be carried out in a way which satisfies the conditions of boundedness and locality. Boundedness means that for each computation there exists a constant bound on the number of objects that can be taken into account in each step. Locality means that only these objects can be modified during each step. Sieg has not been the first person to formulate these conditions. Turing also suggested them, based on psychological considerations (regarding a human computor), while for Gandy they stemmed from considerations regararding mechanical limitations of physical computing devices.

Later Sieg abandoned this idea and focused on another approach to this problem, namely formulating axioms describing properties of computability. He believes we can analyse the notion of intuitive computability and as a result of this analysis formulate axioms. We need to make sure that all known models of computability satisfy these axioms. Then we can prove that any model of computability satisfying these axioms is equivalent to Turing machines (and hence all functions computable in such model are recursive). Thus we can prove Church's thesis. Obviously, we shall consider this sort of proof plausible only if we are convinced that the axioms are correct.

Such argument can help us become more convinced of the truth of Church's thesis, especially if the axioms are detailed and intuitive, which in mathematics is expected as a rule. However, one could argue that it does not allow us to entirely get rid of the chasm between the notion of intuitive computability and the formal axioms.

A different attempt at proving Church's thesis was made by Marcin Mostowski. His proof is carried out within the framework of FM-domains — finite, but potentially infinite models, a concept developed and presented by him in [38], [39], [41] and also together with Konrad Zdanowski in [42]. Mostowski's argument in favour of Church's thesis comes from [40].

The argument rests upon several philosophical assumptions. Whether we accept it as valid, depends whether or not we are willing to adopt the same philosophical perspective.

These are Mostowski's assumptions, as reconstructed in [14]:

1. Ontological assumption: there exist finitely, but potentially infinitely many objects.

2. Semantical assumption: satisfaction and truth relations in finite models are recursive.

3. Epistemological assumption: there exist a recursive enumeration of the FM-domain.

These assumptions need some clarification.

The ontological assumption is related to the distinction between potential and actual infinity introduced by Aristotle in *Physics* [2]. The difference between the two can be explained by invoking the operation of counting. In the process of counting, no matter how far we have progressed, at each step we have only enumerated a finite amount of numbers. However, this process can in principle be continued as far as we wish, therefore it is potentially infinite. Whatever number we reach, we can always continue counting, by saying the name of its successor.[1]

However, when we consider all natural numbers taken as a whole (the set $\mathbb{N}$), we are dealing with actual infinity, since they form an object which actually has infinitely many elements.

Mostowski assumes in his paper that the world we live in is only potentially infinite, not actually. While he claims that, unlike his other assumptions, this one might appear controversial, he believes that it seems reasonable at least with regard to digital computers.

He notices that in a world where computers do not adhere to this principle, Church's thesis might be false. As an example, he mentions a possible world in which time is divisible ad infinitum and a computer can perform each step of its computation in half the time it needed for the previous one. In such a world an infinite number of steps could be performed within a finite amount of time and thus some non-recursive functions would be computable.

Another key notion in Mostowski's assumptions are FM-domains. These are potentially infinite domains understood as increasingly larger sequences of finite models. Let

$$\mathbb{N}_n = (\{0, 1, ..., n-1\}, R_+^{(n)}, R_\times^{(n)}),$$

which is understood as a structure with a domain consisting of natural numbers from 0 to $n-1$, with $R_+^{(n)}$ and $R_\times^{(n)}$ — respectively addition and multiplication, interpreted as relations, restricted to the domain. The domain $FM(\mathbb{N})$ consists of all $\mathbb{N}_n$.

[1] The way of thinking about mathematics based on potential rather than actual infinity was developed by Jan Mycielski in [43].

Usual model-theoretical notions, according to this approach, are considered in the limit. Here is an example of what we mean by this:

Definition 8.3 (M. Mostowski [38]) *The relation $R \subseteq \mathbb{N}^r$ is FM-represented (in $FM(\mathbb{N})$) by a formula $\varphi(x_1, ..., x_r)$ if and only if for all natural numbers $a_1, ..., a_r$ both of the following conditions hold:*

$$R(a_1, ..., a_r) \Leftrightarrow \exists_m \forall_{k \geq m}\ \mathbb{N}^k \models \varphi(a_1, ..., a_r),$$

$$\neg R(a_1, ..., a_r) \Leftrightarrow \exists_m \forall_{k \geq m}\ \mathbb{N}^k \models \neg\varphi(a_1, ..., a_r).$$

If such formula φ exists, we say that R is FM-representable (in $FM(\mathbb{N})$).

Within this theory of potentially infinite universe, Mostowski gives a proof of Church's thesis. He believes, however, that this cannot be considered a purely mathematical proof because it is plausible only if certain philosophical position is accepted. Further details of Mostowski's proofs can be found in the cited articles.

Mostowski's proof was critically analysed by M. Czarnecki, M. T. Godziszewski and D. Kalociński in [14]. They consider the notion of learnability, which — similarly to computability — can be considered from either an intuitive, or a formal perspective. This is how they define intuitive learnability:

Definition 8.4 (Czarnecki, Godziszewski, Kalociński) *A decision problem is intuitively learnable if there is an intuitive algorithm which for each example of the problem runs ad infinitum and produces a finite sequence of answers 'yes' and 'no' such that the last answer in the sequence is correct.*

The idea behind this concept is the following: in the process of learning we can make mistakes, but our knowledge gained this way is correct in the limit (at a certain point we will arrive at a correct answer to our question and then we will stick to it, even if we are not going to know when our answer is correct). The authors give an example of proving a mathematical theorem. In our process of learning whether this theorem is true or false, we begin with the answer 'no'. We analyse all potential proofs, and if there exists a correct one, then at a certain point we are going to consider it and then we shall change our answer to 'yes'.

The situation is reversed if we are testing scientific hypotheses. We start with the answer 'yes' and stick with it until we possibly we come across a counterexample.

A formal concept of learnability was developed by Gold [19] and Putnam [45]. This is Putnam's definition:

Definition 8.5 (Putnam) *Let $A \subseteq \mathbb{N}$. Then A is algorithmically learnable if there is a total computable function $g : \mathbb{N}^2 \to \{0, 1\}$ such that for all $x \in \mathbb{N}$:*

$$\lim_{t\to\infty} g(t, x) = 1 \Leftrightarrow x \in A$$

and

$$\lim_{t\to\infty} g(t, x) = 0 \Leftrightarrow x \notin A.$$

Similarly to Church's thesis, it is possible to formulate Learnability thesis.

Thesis 8.6 (Learnability thesis) *Intuitively learnable sets are exactly those sets that are algrithmically learnable.*

The authors prove that Church's thesis entails Learnability thesis, but they claim that the implication in opposite direction does not hold. They state that the ontological and semantical assumptions of Mostowski's argument taken together are equivalent to a certain version of Learnability thesis. They accept these two assumptions, but they reject the epistemological assumption. They formulate an interpretation of intuitive computability which is consistent with Learnability thesis, but according to which certain intuitively computable sets are not recursive. They argue that the choice of $FM(\mathbb{N})$ as the model to consider is problematic since such choice is equivalent to assuming Church's thesis itself.

8.1.3 Arguments against Church's thesis

While attempts are usually made to prove or in some other way justify Church's thesis, it is also possible that someone could disprove it. In order to do so, it would suffice to give an example of a computable non-recursive function. It is widely believed that such a function does not exist, yet there is already a precedence in form of primitive recursive functions. They are a subset of recursive functions and it is not immediately clear whether it is a

proper subset or not. One could suspect that they contain all computable functions. However, as it turned out, that is not the case. There exist recursive functions which are not primitive recursive. Therefore, it might seem reasonable to suspect that there could exist another, even broader, class of computable functions, containing all recursive functions and more.

Among notable examples of recursive but not primitive recursive functions are the Ackerman function and the Goodstein's function.

Ackermann function can be defined as a function $A : \mathbb{N}^2 \to \mathbb{N}$ in the following way:

$$A_{m,n} = \begin{cases} n+1 & \text{if } m = 0, \\ A(m-1,1) & \text{if } m > 0 \wedge n = 0, \\ A(m-1, A(m, n-1)) & \text{if } m > 0 \wedge n > 0. \end{cases}$$

This is not the only possible way of defining this function, nor the original one. However, this version of Ackermann function is commonly used. While already Hilbert suspected that this function might not be primitive recursive (he speculated so in [24]), it was Ackermann who managed to prove it in [1].

Goodstein's function is more complicated. It was originally introduced by Reuben Goodstein in 1944 in [20]. Kirby and Paris showed in 1982 in [27] that it is a total function (i.e. defined for all natural numbers). Our presentation of this function is based on [9].

For any natural numbers n, b, d, we shall define the depth-d base b representation of n inductively.[2] For $d = 1$ and any n, b, let:

$$n = b^{m_1} n_1 + ... + b^{m_k} n_k,$$

where $m_1 > ... > m_k \geq 0$ and $1 \leq n_i < b$ for each i.

To define depth-$d+1$ base b representation of n, we replace each m_i with its depth-d base b representation. For example, the depth-1 representation of 266 is

$$266 = 2^8 + 2^3 + 2^1,$$

its depth-2 representation is

$$266 = 2^{2^3} + 2^{2+1} + 2,$$

[2]Note that the term 'representation' is used here in a different sense than in the rest of this thesis.

and its depth-3 (or higher) representation is

$$266 = 2^{2^{2+1}} + 2^{2+1} + 2.$$

Note that at a certain point this representation always stabilises, i.e. there exists such d_0 that for all $d \geq d_0$, the depth-d base b representation of n is equal to the depth-d_0 base b representation of n. The depth-d_0 base b representation of n is then called the complete base b representation of n.

The function $R_b : \mathbb{N} \to \mathbb{N}$ (called the change of base function) replaces every b with $b+1$ in the complete base b representation of n. Hence:

$$R_2(266) = 3^{3^{3+1}} + 3^{3+1} + 3.$$

The Goodstein's sequence beginning with n is the sequence $(n)_k$ of natural numbers such that $(n)_1 = n$, and

$$(n)_{k+1} = \begin{cases} R_{k+1}((n)_k) - 1 & \text{if } (n)_k > 0, \\ 0 & \text{if } (n)_k = 0. \end{cases}$$

The Goodstein's function is the function $\mathcal{G} : \mathbb{N} \to \mathbb{N}$ assigning to each natural number n the smallest k such that $(n)_k = 0$. Goodstein proved in [20] that this is a well-defined function.

One could argue that equivalence of all major models of computability speaks in favour of Church's thesis, and that primitive recursive functions do not count as an exception becuse they are simply a subclass of recursive functions. A possible counterargument to this is: perhaps they are all equivalent because they are all based on a certain assumption regarding computability and if we become aware of this assumption, we will be able to construct a better, more general model of computability: a superclass of recursive functions which allows an additional new method of obtaining them, or a new version of Turing machines modified in some way to augment their computational capabilities. It is unclear how probable such a perspective is.

As we have stated earlier, the 'easy' part of Church's thesis can also be problematic. Several possible interpretations of 'intuitive computability' are mentioned in [30]. A function can be computable by a human, a mechanical device, or in some ideal sense. These notions need not be equivalent to each other, and it is not clear which of them both implications in Church's thesis speak of.

In the same source, it is noted that the notion of intuitive computability 'in principle' does not take into account various biological, physical etc. limitations of humans and machines. The humans live only for a certain amount of time which is not sufficient to carry out some long calculations. No mechanical device can continue to work forever due to natural technical imperfections. It is possible to carry on enumerating other limitations of this sort. However, objections of this type are rejected, since, as the author notes, the notion of computability in principle contains a certain idealisation. Physical limitations should not lead us to a conclusion that functions such as addition or multiplication are not computable due to us not being able to calculate the for some very big numbers.

Other attempts at suggesting functions not recursive but possibly computable involve functions computable by machines which increase their speed in an exponential way, thus being able to perform infinitely man steps in a finite amount of time. We have already mentioned such a possibility in the previous subsection.

Another possibility, suggested by Hartmunt Fitz in [16] is that there could exist machines computing non-recursive functions if there are non-recursive physical phenomena which could somehow control these machines. The source of such non-recursivity could, for example, lie in quantum phenomena. However, as is noticed in [30], the problem with such machines is that being non-recursive is an essentially infinitistic property. By observing a certain fragment of reality, we are not able to determine whether it is recursive or non-recursive — both possibilities are always on the table.

8.2 Acceptable and deviant representations

Since natural numbers can be represented in a lot of different ways, various authors have endeavoured to answer the question which of these representations are acceptable. Probably the most notable article discussing this matter has been Stewart Shapiro's 'Acceptable notation' [52].

While Shapiro uses the term 'notations', we shall continue using the word 'representations' throughout this chapter. The reason behind this decision is that we wish to avoid confusion and maintain consistent terminology when discussing different authors' positions. We believe that by doing so we shall in no way misrepresent their views and arguments.

Let us also point out that most articles dealing with this sort of issue

restrict their attention to unambiguous representations of natural numbers, i.e. those according to which each number is represented by exactly one numeral (Definition 2.4). Hence throughout this chapter we shall assume that all representations under consideration are indeed of this type, unless explicitly stated otherwise. While a large part of these considerations is equally valid for ambiguous representations, some technical details of certain arguments might need to be revised to be fully applicable to them.

It seems to be a common position among various authors that representations can be somehow divided into two categories which are often referred to as acceptable and deviant representations. The line of reasoning in support of this view is following:[3]

1. Both mechanical devices and human beings while performing computations deal with physical representations of numbers and not with numbers themselves.

2. These physical representations, whatever their nature, form strings (of symbols).

3. It follows that strictly speaking computability applies only to functions on strings, not to functions on numbers.

4. We can however interpret these functions on strings as somehow representing functions of numbers since the strings represent numbers.

5. There are (infinitely) many ways of interpreting strings (of establishing a correlation between strings and numbers, and as a consequence — between functions on strings and functions on numbers). Each of these interpretations can potentially be a basis for a notion of computability ('is computable in representation X'). Furthermore, we can establish additional notions of computability by quantifying over representations ('is computable in some representation', 'is computable in every representation').

6. We need to find out which of these notions of computability have desirable properties (they are based on acceptable representations) and which of them not (they are based on deviant representations).

Having adopted such a perspective, Shapiro concludes that:

[3]Here it is reconstructed based on Shapiro's article [52].

1. defining computability as 'computability in every (unambiguous) representation' is too narrow because only trivial functions have this property, e.g. if we restrict our attention to unary numerical functions, then only almost constant and almost identity functions satisfy this definition (compare: Theorem 4.2),

2. defining computability as 'computability in some (unambiguous) representation' is too broad since it contains, among others, characteristic functions of all sets.

According to Shapiro, the solution is to define computability as 'computability in every acceptable notation'. This leads us to another problem: formulate a criterion for distinguishing between acceptable and deviant representations.[4]

In order to draw a distinction between acceptable and deviant representations, Shapiro chooses the stroke representation (each number n is represented by a sequence of n strokes) as the basic one. He claims that a person knows or can be taught a representation if they know (or can be taught) an effective procedure (an algorithm) for translating between this representation and the stroke representations (in both directions). Then he shows that if a person knows the numeral for 0 and is able to compute successor, then they know this representation. This is consistent with our Theorem 4.14 if we bear in mind that for Shapiro in every representation $\chi_=$ is by definition computable.

The basis for choosing the stroke notation as the 'standard' for a representation to which all acceptable representations were to be compared, was that 'it is agreed that every normal person knows (or can easily come to know) stroke notation'.

According to Shapiro, to know a representation one needs to be able to read it and write it. A person can read a representation if, given any numeral of this representation, they should (at least in principle) be able to tell what number it denotes, while a person can write a representation if, given any number, they should be able to write a numeral denoting this number in the given representation.

These two approaches provide two different perspectives on what it means to know a representation, even though from a purely extensional perspective

[4]The above argument has been further strengthened by Michael Rescorla by invoking Church's thesis in [48] — his position shall be discussed later in this section.

they lead to the same results. One of them is related to being (mechanically) able to translate between two representations — which is an entirely formal computational approach. The other one speaks of knowing what number is represented. A question could be ask: what does it mean to know what number is being represented? Can we even know such a thing without somehow thinking this number as given in a certain representation — even if the notion 'representation' is potentially understood here in a much broader sense than just a set of finite inscriptions with an interpretation? Even if we think about numbers, we seem to be thinking about a certain representation of them. If we accept this line of reasoning, then 'knowing what number is represented by a numeral' could be synonymous with 'translating a numeral of a certain written representation to a numeral of our preferred language-of-thought representation'.

Shapiro seemed to favour the first approach. He believed it would be problematic to define knowing (reading and writing) a representation in terms of translating between numerals and numbers themselves since it would presuppose some knowledge of numbers independent of any representation. That is why he offered the solution based on the stroke representation. However, this approach was criticised by Rescorla in [48].

Rescorla notices that it is a tempting idea to define an acceptable representation as one in which the successor function is computable. Being able to compute the successor is closely related to the operation of counting and it is a highly desirable property of a representation. However, according to Rescorla, it cannot be a criterion for distinguishing between acceptable and deviant representations since an ambiguous representation (with certain numbers represented by more than one numeral) could still have undesirable properties[5] even despite having computable successor (though his idea of a proof is different, its conclusion bears some similarity to our Theorem 4.15). Therefore he rejects Shapiro's solution.[6] In Rescorla's opinion, defining acceptability of a representation in terms of intertranslatability with another representations is even less convincing since it does not capture the essence

[5]Rescorla gives an example of a representation in which successor is computable, but there is no computable translation from the stroke representation to the representation he considers.

[6]Rescorla notices in a footnote that the computability of both successor and $\chi_=$ is sufficient but he still believes that such analysis is only extensionally valid but does not provide us with a proper explanation of why a certain representation should be considered valid.

of what makes a certain representation acceptable. The question still remains: what feature of the stroke representation makes it a good candidate for a canonical acceptable representation. If we cannot point this feature out, than intertanslatability with any other arbitrarily chosen representation would be just as good for a characteristic property of an acceptable representation — but this is not a conclusion we would like to make.

Rescorla puts a lot of emphasis on distinguishing between explanations valid only extensionally and those valid both extensionally and intensionally. According to him, a good analysis of computability should not only explain which representations are valid but also based on what intrinsic property they can be considered as such. One could argue that this requirement is very vague and subjective.

We can also consider the problem from a different perspective. Regardless of how we want to formulate the criterion, there are two possibilities: either a representation is acceptable if and only if it is isomorphic to the standard representation, or not.

In the first case, one could try to determine what properties of the standard representation, preserved by isomorphisms, are so desirable that we expect every acceptable representation to have them. However, perhaps there is no need to choose between various extensionally equivalent criteria.

We believe most of these criteria offer some interesting philosophical insight into representations (computability of some key functions or sets, translatability to another established representations, etc.) , however any attempts at choosing just one of them as expressing the very essence of 'computability' are futile. Different criteria might point out at different perspectives regarding representations.[7] The computability of successor and $\chi_=$ constitute a good criterion since they are probably the most basic way of using numbers (counting objects and determining whether two sets of objects are of equal cardinality). The process of counting has been invoked as the origin of the concept of number (for example by L. Kronecker in [31]). It is

[7]Some of them — even if extensionally equivalent — express properties we would normally not think of as being key to a representation, e.g. a property demonstrated in Theorem 4.21.This theorem states that a representation is isomorphic to the standard representation if and only if both f and $\chi_=$ are computable in it, where f is any non-identity strictly increasing function computable also in the standard representation. This theorem has been proved in order to to show that the computability of successor and $\chi_=$, as intuitive as they seem, can easily be replaced by another less obvious criterion — and both these criteria are satisfied by exactly the same class of representations.

fortunate that these two assumptions taken together implicate all the other desirable properties of a representation, as has been proved (Theorem 4.14).

Furthermore, one could argue that computability is largely a pragmatic notion. People usually compute values of different functions (regardless if they use any devices for doing so or not) in order to gain knowledge which they for whatever reason consider important or interesting. From such a perspective, a computation is valid only if its result is given in such a form that it increases our knowledge in a meaningful way and preferably can be further utilised.

For example, consider the halting problem which is not computable in the standard representation but is computable in many non-standard representations. Why, when considering this problem, do we choose to favour the standard representation over those others? The answer could be that computing the halting problem would only be interesting if there was a way to establish an understandable connection between algorithms and numbers (or numerals) used to encode them. We want to be able, for each algorithm, to determine what number is its code, and for each number to know what algorithm it encodes. Only then would computing the halting problem give us actual knowledge about these algorithms. Since we are not able to do it, for all practical purposes the halting problem is not computable.

The question appears to be more interesting in the second case i.e. if acceptability of a representation is not extensionally equivalent to being isomorphic to a standard representation. Then there are two mutually non-exclusive possibilities: either there exists an unacceptable representation isomorphic to the standard one, or there exists an acceptable representation not isomorphic to the standard one.

If there exists an unacceptable representation isomorphic to the standard one, then the unacceptability of such a representation would probably need to be justified by some pragmatical reasons, for example it would be too slow to compute certain functions in it. If a researcher's investigations are motivated mainly by philosophical reasons, such as examining the relation between Church's thesis and various representations, then it seems that there should be no reason to even distinguish between various isomorphic representations: either all of them should be acceptable, or all of them should be deviant.

An acceptable representation not isomorphic to the standard one is very hard to imagine. It appears to be a general consensus that such a possibility should be rejected and the authors debating the topic differ with regard to justification of this stance. Various possible justifications have been discussed

in this section.

This is an understandable position. If one defines 'computability' in terms of 'computability in all acceptable representations', then all such representations need to be equivalent and thus — if we assume computability of $\chi_=$ — also isomorphic (due to Theorem 3.12). If we additionally assume that the standard representation is acceptable, then it follows that every acceptable representation needs to be isomorphic to the standard representation.[8] This is especially clear in the context of considerations contained in the following section.

8.3 Multitude of representations and Church's thesis

According to Church's thesis, computable functions are exactly those functions which are computable by a certain Turing machine. However, as noticed by Rescorla in [48], the computability of a given function is usually dependent on the choice of a representation. Church's thesis presupposes the use of either the standard representation, or any representation isomorphic to it — this assumption is considered so obvious that it does not even need to be questioned. However, adopting different representations leads to different notions of computability. Therefore Church's thesis cannot be maintained without specifying a representation (or representations).

Furthermore, should we allow 'deviant' representations, then some uncomputable functions would become computable — and that is not a desirable conclusion either.

Rescorla examines Turing's argument in favour of Church's thesis (following the interpretation given by Gandy and Sieg). According to Sieg, Turing formulated the following constraints to which all possible mechanical computations are subject:

1. The behaviour of the computing agent is fully determined by the configuration of symbols he observes and by his own internal state (in case of a human it can be interpreted as his state of mind).

[8]The implication in the opposite direction does not follow from these assumptions though.

2. There is a fixed finite limit on the number of symbolic configurations the computing agent is capable of recognising.

3. There is a fixed finite limit on the number of internal states which need to be taken into account.

4. Only elements of observed configurations of symbols can be changed.

5. There is a constant bound on how far each new observed element can be from immediately previously observed elements.

Turing believed that these constraints hold for any mechanical computing performed by human agents and that the mechanical activity of any human satisfying these conditions could be mimicked by a Turing machine.

Rescorla formulates a series of criticisms towards this analysis. According to him:

1. some of constraints imposed by Turing might hold for all human computing agents but it is not certain if they hold for all computing agents in general (especially the second and the forth restraint seemed dubious to Rescorla),

2. Turing's analysis does not concern all of human cognition, in particular it does not rule out the possibility of existence of non-mechanical means available to humans of determining values of certain functions,

3. the notion of a 'state of mind' is not well-defined,

4. even if we put all those doubts aside, Turing's analysis still only proves at best that every intuitively computable function is computable by a certain Turing machine in a certain representation, not necessarily in any representation that is acceptable.[9]

The criticism that concerns us here most is the last one. According to Rescorla, a computation carried out on numerals of a deviant representation could still satisfy all of Turing's conditions, thus should be treated as a valid computation (if we were to accept Turing's analysis without any reservations) and a function computed this way — due to Church's thesis — would be (intuitively) computable. Yet Rescorla does not want to recognise such a

[9]In Rescorla's terminology, representations are called 'semantics'.

function as being computable. He argues in favour of adding to Turing's list of requirements the Computability Constraint:

> the symbols that the computor manipulates bear a computable semantic relation to the numbers they denote.

Using our terminology: a representation (S, σ) is acceptable (non-deviant) only if σ is computable. Rescorla explains that computability of a semantic is understood as intuitive computability which makes use only of mechanical means.

Another strategy of dealing with the same problem is employed by Jack Copeland and Diane Proudfoot in [12]. They divide deviant representations (called by them 'encodings') into deviant input encodings and deviant output encodings. While they do not explicitly define what exctly is meant by an encoding, it appears to be a broader term than representation understood as in this thesis.

Copeland and Proudfoot reason in terms of Turing machines. Computation of a function is a mechanical procedure carried out by a Turing machine on which input and output are encoded in a certain way, not necessarily identical in both cases. Based on examples given in their paper, we try to describe how their encodings differ from our representations. The word 'inscriptions' is going to be used here instead of 'numerals' since it is more versatile and more suitable in this context.

1. The set of all inscriptions does not need to be computable.

2. Inscriptions given as the intput can be interpreted in a different way than inscriptions returned as the output.

3. Inscriptions do not always represent numbers, they can also represent other objects, such as logical values.

It is worth noting that while various authors usually define representation in a much stricter way than we do, Copeland and Proudfoot went in the opposite direction and adopted a very broad definition.

Let us consider the halting problem, which is not computable in the standard representation. Copeland and Proudfoot provide examples of deviant encodings (both deviant input and output encodings) relative to which this problem is solvable.

The deviant input encoding is the following: let us enumerate all Turing machines. Let n be the index of a machine. We want to check whether this machine halts. If it halts, let the input be $2n$, otherwise let the input be $2n + 1$. If the input is even, print $HALTS$ as the output, otherwise print $NOT - HALTS$.

Let us observe that this is either not a valid representation[10] according to our definition or not a valid algorithm. The reason for this is that the set of all possible inputs is not computable — for every natural number n, only one of the numerals $2n$ and $2n + 1$ constitutes a valid input for the algorithm (or Turing machine) computing the halting problem. If we assume that numerals accepted by such an algorithm are all numerals of the representation, then the set of all numerals of the representation is not computable, thus it is not a valid representation. If, on the other hand, these numerals only constitute a proper subset of the set of all numerals, then it is not a valid algorithm because an algorithm needs to work for any numeral given as input.

Copeland and Proudfoot claim that Turing himself has already solved the problem of deviant input encodings by requiring that the tape of the machine at the beginning of the computation should be blank. It implies that other than the input, no additional information can be given. The deviant encoding suggested above does not comply with this rule since it contains 'hidden' information regarding the answer to the question. Copeland and Proudfoot illustrate it in the following way:

> You can give the correct answer to any Yes/No question that I ask you, if it is arranged in advance that I will wink at you (as I ask the question) if and only if the correct answer is Yes.

According to them, Turing defined computability in terms of computable sequences. The machine starts with a blank tape and prints consecutive numerals as the output. For example, following this approach, computing the function $f(n) = 2^n$ for all non-negative integers would consists in generating a sequence of numerals representing the following numbers: $1, 2, 4, 8, 16, ...$

[10]The term 'valid representation' refers simply to a correct representatiom, i.e. an ordered pair (S, σ) which satisfies all the conditions contained in the definition of a representation. This is a purely mathematical notion, possibly unlike 'acceptable representation' which is the opposite of a 'deviant representation'. Both 'acceptable representation' and 'deviant representation' are concepts with a philosophical component and their meaning is one of central topics in this chapter.

The argument based on blank tape restriction has also been criticised by Rescorla in [49]. He claims that such approach is too restrictive since it does not take into account all possible computations. Furthermore, it deviates too much from usual computational practices, even as basic as algorithm for multiplication of natural numbers taught at primary schools, where rather than computing a whole sequence, we only need to calculate the value of the function for argument(s) given as input.

One could further argue that computability interpreted this way could be hard to generalise to multivariable functions or functions defined on objects other than natural numbers.

The case of a deviant output encoding seems even more extreme. Consider an enumeration of all Turing machines. The Mirabilis Turing machine prints decimal numerals, one by one. If the nth Turing machine halts, then the nth inscription in this sequence means 'The nth Turing machine halts', otherwise it means 'The nth Turing machine does not hold'. Such a machine is capable of computing whether the nth Turing machine halts, for every natural number n. Obviously, this is considered a deviant encoding.

It seems that this example cannot be translated into our terminology because:

1. it would require a representation to have infinitely many inscriptions representing each logical value; however, we assume that logical values are unambiguously represented by inscriptions $TRUE$ and $FALSE$,

2. this computation does not have any input which is not allowed by our concept of computability within a representation.

Copeland and Proudfoot believe that deviant output encodings can be ruled out by following Turing's approach as well. In [62], Turing defines a computable function in terms of a concrete encoding, which is called Turing-unary by Copeland and Proudfoot, namely all terms of the sequence are expressed in unary notation and are separated by a 0. This condition allows us to rule out all deviant output encodings.

In his reply to this, Rescorla repeated his old argument, namely that while unary notation is acceptable, any proper analysis of computability, needs to explain fundamental reasons why we choose this notation and not any other one [49].

Copeland and Proudfoot can be interpreted as trying to stay close to Turing's original concept and to find solution to various philosophical prob-

lems in his own writings. While this is a valid position, it might not be sufficient for those who wish to consider a wider class of computations from a philosophical perspective, like Rescorla does.

8.4 Representations and Turing's analysis of computability

It has been argued that Turing's analysis of computability is circular, which is caused by existence of a lot of different computationally non-equivalent representations. This argument has been discussed by Rescorla in [48].

According to Rescorla, a good analysis of any concept needs to satisfy the following conditions:

1. It should be extensionally adequate, i.e. any object should satisfy this definition if and only if it falls under the analysed concept,
2. It should be non-circular.
3. It should capture the meaning of the analysed concept.[11]

According to Rescorla, Turing's analysis of computability satisfies the first condition and — only with regard to string-theoretic functions — the second condition. It fails, in his opinion to satisfy the second condition with regard to number-theoretic functions and to satisfy the third condition. This is because it does not give us any convincing reason to choose one representation over any other, nor does it provide us with means to rule out deviant representations.

As has been discussed in the previous section, Rescorla believed that for Turing's analysis to be accurate, a restriction needs to be introduced as to which representations are considered acceptable and which are not. Rescorla suggested accepting only those representations where correlation between numerals and numbers is computable, however he believed this approach brings its own share of difficulties. Combined with Turing's analysis and Church's thesis, it would entail that 'a function is computable if and only if it its value can be determined by a Turing machine according to certain

[11] This condition is obviously the most unclear and vague one.

computable semantics' — but such analysis would suffer from circularity, thus it does not satisfy the second condition.

The alternative would be to choose a canonical representation (for example unary or decimal) and define computability as computability in that chosen representation (this is what Shapiro did), in which case the analysis would fail to satisfy the third condition. This is because according to Rescorla arbitrarily choosing a representation (or even choosing it based on somebody's mathematical practice) does not capture the meaning what 'computability' really means.

Rescorla analyses different possible ways of addressing this issue but he rejects them. Among others, he discusses the following possibilities:

1. The Translation Constraint: a representation is acceptable if there exist an intuitively computable translation between this representation and the (standard) decimal representation. Rescorla rejects it because, despite being extensionally correct, it fails to satisfy the third condition and therefore cannot be considered a proper analysis of the concept. It does not explain why intertranslatability with the (standard) decimal representation is more desirable than intertranslatability with any other representation.

2. The Successor Constraint: a representation is acceptable if successor is computable in this representation. As Rescorla noticed, if we allow non-injective representations, the computability of $\chi_=$ needs to be included in this condition as well. But even with this additional requirement, according to Rescorla it is only extensionally adequate and fails to satisfy the third condition.

Another possibility considered by Rescorla is a purely syntactic conception of computation, according to which all computations are performed on strings and not on numbers. If we decide to follow this path, then the problem of representations disappears. However, he discards this solution and arguments suggested in support of it for the following reasons:

1. It is inconsistent with established mathematical practise which treats computability as a notion primarily related to numbers.

2. It is unintuitive and instead of solving the problem of drawing a distinction between acceptable and deviant representations, it chooses to

dispose of the problem by eliminating the notion of computability on numbers themselves.

3. Even if humans and devices during computations deal directly with strings of symbols rather than numbers, it does not follow as a legitimate conclusion that computability is only a string-theoretic notion. At best one can draw a conclusion that computations on functions on numbers are performed via computations of functions on strings.

4. The purely syntactic concept of computation cannot be established on the basis of any chosen notation system, interpreted as an ω-sequence, being isomorphic to the natural numbers. Rescorla considers the existence of such isomorphism to be irrelevant to this debate since it fails to explain why we possess an intuitive notion of numerical computability.

5. Rescorla also mentions that some people could prefer purely syntactic approach for historical reasons (he believes such interpretation is sometimes assigned to works of Turing and Gödel), but he concludes that historical reasons are not a good enough reason to assume one notion of computability over another. He also emphasises that such approach was not unanimously accepted, for example Kleene's and Post's intentions clearly went beyond the realm of purely syntactic computations.

Unfortunately, Rescorla does not offer a better positive solution. He acknowledges advancement made in computation theory thanks to syntactic approach, yet, he believes, it would be a mistake to focus entirely on this perspective, while disregarding semantics.

It could be argued that perhaps Rescorla was too hasty in rejecting characterising acceptable representations based on computability of successor and identity. These conditions seems to capture the essence of what are the most basic operations we want and need to perform on natural numbers. We emphasised their role in section 8.2.

8.5 Similarities to Tennenbaum's theorem

Similar discussions have been held with regard to Tennenbaum's theorem which is often utilised to draw a line between intended and not intended models of arithmetic. This theorem faces analogous criticism of its alleged

circularity, regarding notion of natural numbers. We have decided to include the summary of these discussions since a lot of arguments presented in this debate could be equally valid for the discussion included in the previous section. Furthermore, the debate regarding Tennenbaum's theorem has already been referred to in this context (compare: [46]).

The reconstruction of the theorem is based on [26].

Theorem 8.7 (Tennenbaum's theorem [60]) *Let*

$$\mathbb{M} = (M, +^{\mathbb{M}}, \cdot^{\mathbb{M}}, <^{\mathbb{M}}, 0^{\mathbb{M}}, 1^{\mathbb{M}})$$

be a countable non-standard model of PA. Then, neither $+^{\mathbb{M}}$ *nor* $\cdot^{\mathbb{M}}$ *is recursive on* $\mathbb{M}$*.*

A structure $\mathbb{M} = (M, +^{\mathbb{M}}, \cdot^{\mathbb{M}}, <^{\mathbb{M}}, 0^{\mathbb{M}}, 1^{\mathbb{M}})$ is recursive if there exists an injective function $f : \mathbb{N} \to \mathbb{M}$, identifying $\mathbb{N}$ with M, such that $+^{\mathbb{M}}, \cdot^{\mathbb{M}}, <^{\mathbb{M}}$ are all recursive on $\mathbb{N}$. It is important to emphasise that these functions and the ordering on the initial segment of $\mathbb{M}$ do not need to correspond to the normal functions and ordering on $\mathbb{N}$.

Tennenbaum's theorem is still valid for theories of arithmetic much weaker than PA.

Tennenbaum's theorem has been used as an argument in debates regarding distinguishing intended and non-intended models of arithmetic. A theory can be instantiated by infinitely many different models. While it is usually accepted that isomorphic models for all our purposes can be treated as identical, often various models of the same theory are not even isomorphic. Due to Skolem-Löwenheim theorem we are aware of existence of non-standard models of first-order theories of arithmetic, such as Peano arithmetic.

Yet at the same time we would like to be able to say that arithmetic is about natural numbers. Therefore it would be desirable to have tools at our disposal which would enable us to somehow distinguish between intended and unintended interpretations of arithmetic. In other words, we wish to know which structures satisfying axioms of Peano arithmetic can be considered as 'true' natural numbers.

Paul Benacerraf claimed in [4] that such a structure should be an ω-sequence and furthermore the ordering of the structure should be recursive, later in [5] he dropped the second requirement and decided that it is enough to assume that the model should have order type ω.

Another method proposed by Shapiro (in [53] and [54]) was to resort to second-order logic. This is because first-order logic is too weak to describe ω-sequence.

However, in their article [22], Volker Halabach and Leon Horsten reject all these attempts and argue that the intended models of Peano arithmetic can be specified by referring to Tennenbaum's theorem.

They notice that defining (a model of) natural numbers by appealing to ω order type is circular since to understand what ω order is, we already need to understand the notion of natural numbers. Resorting to second-order logic is even more problematic since we would need to make use of the notion of power set of natural numbers which is even harder than the notion of natural numbers themselves.

In Halbach and Horsten's view, Tenenbaum theorem provides a good method of determining which models of natural numbers are valid since it speaks of core properties of natural numbers, such as being able to carry out calculations of addition and multiplication and at the same time it manages to avoid circularity of other attempts at a definition.

While the authors notice a potential source of a different circularity in their solution, they believe they have managed to overcome it. This alleged circularity is the following: Tennenbaum's theorem makes use of the notion of recursive functions. However, this notion is defined for functions on natural numbers and so in order to understand what recursive functions are, we already need to have natural numbers defined.

Halbach and Horsten reject this objection on the basis of a distinction between theoretical and practical computability, following an argument made by [15]. The usual definition of recursive functions, which presupposes understanding of what natural numbers are, constitutes the theoretical notion of computability. We rely on this notion in the proof of Tennenbaum's theorem. However, in our philosophical consideration based on already proven Tennenbaum's theorem, we speak of the practical notion of computability. This notion is based on an intuitive understanding of an effective procedure and does not require any in-depth analysis of what natural numbers are. The gap between these two notions can be bridged by the use of Church's thesis (according to terminology from section 8.1, it is an essential use of the thesis).

Furthermore, Halbach and Horsten believe that in these considerations we can avoid mentioning natural numbers altogether. They suggest we need to assume that axioms of Peano arithmetic do not speak of numbers but

rather of notation systems (which are similar to representations discussed in this thesis). Thus, they claim that intended models are notation systems with recursive operations satisfying the Peano axioms.

A similar line of reasoning has been adopted by Paula Quinon and Konrad Zdanowski in [47], however they focus more on computational experiences with natural numbers rather than on a formal theory such as Peano arithmetic. They assume computability of basic arithmetical operations, the principle of induction and psychological version of Church's thesis (any property that human can compute can also be computed by Turing machines) and via Tennenbaum's theorem they conclude that the intended models for arithmetic are recursive models with ω-type ordering.

The argument based on Tennenbaum's theorem is criticised by Tim Button and Peter Smith in [8] — but for being unconvincing, not unsound. They reject Halbach and Horsten's reasoning which is based on distinction between theoretical and practical notions of computability. They believe that the concept of practical computability is just as problematic as that of a recursive function. It is related to what we are able to do 'in principle', i.e. in an arbitrary finite number of steps, given enough time and memory — but to mathematically establish each of these notions we would need to already know what natural numbers are.

The authors believe that an argument appealing to Tennenbaum's theorem is unconvincing because if someone is sceptical enough to have doubts about natural numbers, no amount of additional explanations regarding model theory is going to convince them. However, this needs not worry us since, according to Burton and Smith, the problem at hand is not a genuine one. No mathematical discourse happens in an entirely new language free of interpretation, we always employ some theory from the start. Therefore, the authors conclude,

> philosophical problems which are supposedly generated by mathematical results can rarely be tackled by offering more mathematics.

There is a parallel between philosophical discussions regarding Church's thesis (in context of representations) and Tennenbaum's theorem. The former is used to establish the notion of computability, the latter — of natural numbers. Both these notions are crucial to mathematics, and both these arguments have had to face similar criticism regarding alleged circularity in

their appeal to the notion of computability. In addition to that, in both cases, in order to refute such criticism, the distinction between practical (intuitive) and theoretical notion of computability has been invoked.

Furthermore, it seems that the argument raised by Halbach and Horsten could also be applied to discussions surrounding Church's thesis and various representations of natural numbers. Namely, even if there is some circularity in the interpretation of Church's thesis, related to the idea of acceptable representations, there is no need for concern. Some notions always need to be accepted as understandable without further explications since mathematics always utilises some already established terminology, there is no escaping that.

Similarities between both discussions have led us to refer to Tennenbaum's theorem in this chapter. However, neither of the problems has been solved in an unanimously accepted way. It is worth noting that the subject of non-standard models of Peano arithmetic is much more popular among researchers than that of representations of numbers, therefore also much more attention is being paid to its philosophical aspects.

9 Conclusions

9.1 Summary of our results

In this thesis we have considered computational properties of various representations of natural numbers. Each chapter dealt with different aspects of this subject. Here we would like to formulate some conclusions from our investigations.

1. Usual mathematical notions can often be defined in many different ways within the representation-based framework. In Chapter 2 we provided two different definitions of computability which in general case are not equivalent to each other. In Chapter 3 we considered what it means that two representations are similar and, again, it turns out that there are several different ways of doing this. Some concepts do not translate well to this new perspective, for example Turing reducibility of functions, considered in Chapter 7.

2. The source of this problem lies in the ambiguity of representations. The standard notion of computability does not need to take into account that there are many equally valid answers to a certain question. For example, if the value of a certain function is represented in a given

representation by two different numerals, α and β, then they are both equally valid answers to the question about the value of that function. As it turns out, it is often significant which of several interpretations of a notion we choose — they have different properties.

3. If we limit our attention to unambiguous representations, then this difficulty disappears and all these various definitions become equivalent. We have observed several cases of this phenomenon:

   - equivalence of both notions of computability (Chapter 2),
   - equivalence of various criteria for similarity of representations (Chapter 3) and for comparing representations (Chapter 6),
   - equivalence of various definitions of Turing reducibility of functions (while it has not been explicitly stated as a theorem in Chapter 7, it is obvious).

4. The assumption of unambiguity of a representation in the previous point is too strong. It suffices to assume that the relation of identity is computable on the set of all numerals, i.e. that there exists an algorithm which for any two numerals α and β decides whether they represent the same number or not. Representations in which that is not the case, exhibit very little regularity in their computational properties. On the other hand, every representation with computable identity is isomorphic to a certain unambiguous representation (Theorem 3.7) — which for all our purposes means that these representations have identical computational properties.[1] Hence, Shapiro's restriction to unambiguous representations, whether consciously or not, was justified in the following sense: every representation with reasonable computational properties is isomorphic (i.e. practically identical) to a certain unambiguous representation.

5. We considered computability of various functions, mostly in Chapter 4. We have concluded that in general case functions are computationally largely independent from each other, i.e. for two functions f and g usually there is a representation in which one of them is computable and the other is not. It seems that computability of one function implies

[1] Some properties related to Turing reducibility are an exception here, but they are only a marginal topic in this thesis.

computability of the other only if the latter can be obtained from the former using some very simple methods (example: Theorem 4.25). In section 9.2 we discuss possible ways of generalising these results in more detail.

6. In representations with decidable identity, the situation changes and computability of various functions manifests much more regular properties. This has been discussed in Chapter 5 in which we introduced notions of normal representations, canonical representations and generating permutations. These notions are very helpful at describing such representations and proving their computational properties. It turns out that such representations are very similar to each other in various ways described in that chapter.

7. It is an interesting result that the standard decimal representation of $\mathbb{N}$ cannot be improved, i.e. there is no representation which allows us to compute all the usual computable functions and even more in addition to that (Theorem 6.14).

We believe that the choice of a representation is usually a pragmatic one. For most purposes, the standard decimal representation or an isomorphic one are probably most useful. Regardless, it is important to study properties of various representations for theoretical reasons.

We consider our results to be an important step in conceptual analysis of computability-theoretical notions. Putting them in a new, wider perspective helps us understand them better. For example, computability of functions such as successor, addition, multiplication and exponentiation is often taken as obvious, inseparable from each other. By considering these notions in various representations we can better understand relations between them.

9.2 Further questions for investigation

While we believe we have contributed significant new results to the research of representations, further questions arise which need to be investigated. In this section we wish to briefly present some of them.

One of the most important issues we considered here have been relations between computability of various functions. We believe it is possible to give our results a more generalised form.

Suppose that we have a representation (S, σ) of $\mathbb{N}$ about which we assume that all sets from a countable family $\mathcal{F}$ and all relations from a countable family $\mathcal{R}$ are computable in it. What other functions and relations must be computable in (S, σ)? We would like to have a general theorem which answers this question.

For the sake of this section, let us divide this problem into two separate questions — one regarding functions, and the other one regarding relations. The following theorems are straightforward:

Theorem 9.1 *Let (S, σ) be a representation of $\mathbb{N}$ in which all functions from a countable family $\mathcal{F}$ are computable. Then the family $\mathcal{G}$ of all functions computable in (S, σ):*

1. *contains all constant functions and projections,*
2. *contains all functions from $\mathcal{F}$,*
3. *is closed under composition of functions,*
4. *is closed under changing order of arguments of a function, i.e. if $f : \mathbb{N}^n \to \mathbb{N}$ belongs to $\mathcal{G}$, and $\pi : \{1, ..., n\} \to \{1, ..., n\}$ is a permutation, then the function g defined as follows, for any $x_1, ..., x_n$, also belongs to $\mathcal{G}$:*
$$g(x_1, ..., x_n) = f(\pi(x_1), ..., \pi(x_n)).$$
5. *is closed under adding new irrelevant arguments, i.e. if $f : \mathbb{N}^n \to \mathbb{N}$ belongs to $\mathcal{G}$, then the function $g : \mathbb{N}^{n+1} \to \mathbb{N}$ defined as follows:*
$$g(x_1, .., x_n, x_{n+1}) = f(x_1, ..., x_n),$$
for any $x_1, ..., x_n, x_{n+1}$, also belongs to $\mathcal{G}$.

Theorem 9.2 *Let (S, σ) be a representation of $\mathbb{N}$ in which all relations from a countable family $\mathcal{R}$ are computable. Then the family $\mathcal{T}$ of all relations computable in (S, σ):*

1. *contains the empty set and all sets $\mathbb{N}^n$, for $n \in \mathbb{N}$,*
2. *contains all relations from $\mathcal{R}$,*
3. *is closed under Boolean combinations,*

4. *is closed under changing order of arguments, i.e. if* $R \subseteq \mathbb{N}^n$ *belongs to* $\mathcal{T}$*, and* $\pi : \{1, ..., n\} \to \{1, ..., n\}$ *is a permutation, then the relation* T *defined as follows, for any* $x_1, ..., x_n$*, also belongs to* $\mathcal{T}$*:*

$$(x_1, ..., x_n) \in T \Leftrightarrow (\pi(x_1), ..., \pi(x_n)) \in R.$$

5. *is closed under substitution, i.e. if* $R \subseteq \mathbb{N}^n$ *belongs to* $\mathcal{T}$*, then the relation* $T \subseteq \mathbb{N}^{n-1}$ *defined as follows:*

$$(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n) \in T \Leftrightarrow (x_1, ..., x_{i-1}, a, x_{i+1}, ..., x_n) \in R,$$

for any $x_1, ..., x_n, a$*, also belongs to* $\mathcal{T}$*.*

6. *is closed under adding new irrelevant arguments, i.e. if* $R \subseteq \mathbb{N}^n$ *belongs to* $\mathcal{T}$*, then the relation* $T \subseteq \mathbb{N}^{n+1}$ *defined as follows:*

$$(x_1, ..., x_n, x_{n+1}) \in T \Leftrightarrow (x_1, ..., x_n) \in R,$$

for any $x_1, ..., x_n, x_{n+1}$*, also belongs to* $\mathcal{T}$*.*

We have tried to keep the above lists minimal. For this reason, we have not included the following conditions which can be deduced from the above theorems.

1. The family $\mathcal{G}$ is closed under substitutions because a substitution of a constant for a variable in a function f is a composition of f with a constant function.

2. The families $\mathcal{G}$ and $\mathcal{T}$ are closed under removing irrelevant arguments because we can obtain a new function, set or relation (without an irrelevant argument) by substituting any constant for this argument.

3. The family $\mathcal{T}$ is closed under concatenation of relations, i.e. if $R_1 \subseteq \mathbb{N}^k$ and $R_1 \subseteq \mathbb{N}^l$ are both in $\mathcal{T}$, then relation $R \subseteq \mathbb{N}^{k+l}$ defined as follows, for any $x_1, ..., x_k, y_1, ..., y_l$ also is:

$$(x_1, ..., x_k, y_1, ..., y_l) \in R \Leftrightarrow (x_1, ..., x_k) \in R_1 \wedge (y_1, ..., y_l) \in R_2.$$

This is because we can first add l new irrelevant variables to R_1 on the right and k new irrelevant variables to R_2 on the left, thus obtaining respectively R_1' and R_2'. Then let $R = R_1' \cap R_2'$. A similar argument can be

used if we replace conjunction in the above definition with any other Boolean connective.

What is not obvious, is whether families satisfying all the conditions listed above are minimal.

Hypothesis 9.3 *Let $\mathcal{F}$ be a countable (finite or infinite) family of functions on natural numbers and $\mathcal{G}$ — the family of all functions computable in every representation of $\mathbb{N}$ in which all functions from $\mathcal{F}$ are computable. Then $\mathcal{G}$ is the smallest family which contains all constant functions and projections and all functions from $\mathcal{F}$ and is closed under composition of functions, changing order of arguments and adding irrelevant new arguments.*

Hypothesis 9.4 *Let $\mathcal{R}$ be a countable (finite or infinite) family of relations on natural numbers and $\mathcal{T}$ — the family of all relations computable in every representation of $\mathbb{N}$ in which all relations from $\mathcal{R}$ are computable. Then $\mathcal{T}$ is the smallest family which contains the empty set, all sets $\mathbb{N}^n$ and all relations from $\mathcal{R}$ and is closed under Boolean combinations, changing order of arguments, substitutions and adding irrelevant new arguments.*

While we have not proved these hypotheses thus far, we suspect that they are true. This is because, as we have observed in our research, computability of any two functions or relations is usually independent from each other, unless there is a really good (and often obvious) reason for computability of one of them to imply the computability of the other. Such implication often has it roots in existence of an easy way to transform one function (or relation, or a set thereof) into another function (or relation). These are some examples from this thesis:

1. Computability of addition implies computability of successor (Theorem 4.16). Note that successor can be obtained from addition by substitution, i.e composition of addition and a constant function.

2. If we assume computability of a certain finite non-empty family of subsets of $\mathbb{N}$, then the only subsets of $\mathbb{N}$ whose computability is guaranteed, are their Boolean combinations (Theorem 4.25). Note that this, in particular, gives us computability of empty and full sets, by applying the following Boolean combinations: $\emptyset = A \setminus A$ and $\mathbb{N} = A \cup (\mathbb{N} \setminus A)$, for any A belonging to this family. All the other conditions from Theorem 9.2 are not relevant since we are only concerned here with unary relations.

3. Theorem 4.5 can be treated as a special case of Hypothesis 9.3, where no assumptions are made regarding computability of any functions, i.e. $\mathcal{F} = \emptyset$.

Should the above hypotheses turn out to be false, we still suspect that they could be modified by adding finitely many new ways of obtaining additional functions. We believe that each of these methods would need to be very simple and finitistic in nature.

Alternatively, perhaps one could limit these hypotheses only to situations when $\mathcal{F}$ and $\mathcal{R}$ are finite.

Another question to consider are relations between various criteria for similarity of representations. Not all of them have been established in Chapter 3. In particular, we wish to know if, in general case, equivalence of two representations implies that they are χ-equivalent or isomorphic. If we answer this question, then our picture of relations between this criteria will be complete.

We would also like to investigate more thoroughly into representations of other types of objects than just natural numbers. Elements of non-standard models of Peano arithmetic seem to be an interesting direction to pursue, especially in the context of Tennenbaum's theorem, which could be interpreted in the following way:

Theorem 9.5 (Tennenbaum's theorem, alternative formulation) *Let M be the universe of a certain countable non-standard model of PA, and (S, σ) be a representation of M. If $\chi_=$ is computable in (S, σ), then neither addition, nor multiplication is.*

Proof.

Let (S, σ) be a representation of M. Then, due to Theorem 3.7, there is an unambiguous representation (T, τ) of M which is isomorphic to (S, σ). Note that $(T, +^\tau, \cdot^\tau)$ is a non-standard model of PA. Due to the original formulation of Tennenbaum's theorem, $+^\tau$ and $\cdot^\tau$ are not computable. Hence, $+^\sigma$ and $\cdot^\sigma$ are not computable, since (S, σ) is isomorphic to (T, τ) and, as a result, also equivalent.

■

Some research into representations of finite sets could also prove interesting. The results obtained by M.T. Godziszewski and J. D. Hamkins in [18] might be valuable in such research.

This is certainly not an exhaustive list of all questions that remain to be answered in this area. There has been relatively little research going on in this subject and we believe there is a lot of work to be done regarding investigation of representations of numbers.

Bibliography

[1] Wilhelm Ackermann. Zum Hilbertschen Aufbau der reellen Zahlen. *Mathematische Annalen*, 99:118–133, 1928.

[2] Aristotle. *Physics*. University of Nebraska Press, Lincoln, 1961.

[3] Eric Temple Bell. *Men of Mathematics*. Simon & Schuster, New York, 1986.

[4] Paul Benacerraf. What Numbers Could Not Be. *Philosophical Review*, 74(1):47–73, 1965.

[5] Paul Benacerraf. Recantation or Any Old ω-sequence Would Do After All. *Philosophia Mathematica*, 4(2):184–189, 1996.

[6] George Boole. *Mathematical Analysis of Logic, Being an Essay Towards a Calculus of Deductive Reasoning*. Macmillan, Barclay, & Macmillan, Cambridge, 1847.

[7] George Boole. *An Investigation of The Laws of Thought on Which Are Founded the Mathematical Theories of Logic and Probabilities*. Macmillan, London, 1854.

[8] Tim Button and Peter Smith. The Philosophical Significance of Tennenbaum's Theorem. *Philosophia Mathematica*, 20(1):114–121, 2012.

[9] Andrés Eduardo Caicedo. Goodstein's Function. *Revista Colombiana de Matemáticas*, 41(2):381–391, 2007.

[10] Alonzo Church. A Note on the Entscheidungsproblem. *Journal of Symbolic Logic*, 1(1):40–41, 03 1936.

[11] Alonzo Church. The Constructive Second Number Class. *Bulletin of the American Mathematical Society*, 44:224–232, 1938.

[12] Brian Jack Copeland and Diane Proudfoot. Deviant Encodings and Turing's Analysis of Computability. *Studies in History and Philosophy of Science Part A*, 41(3):247–252, 2010.

[13] Louis Couturat. *Opuscules and fragments inédits de Leibniz. Extraits des manuscrits de la Bibliothèque royale de Hanovre.* Alcan, Paris, 1903. Reprinted by G. Olms Verlagsbuchhandlung (1961).

[14] Marek Czarnecki, Michał Tomasz Godziszewski, and Dariusz Kalociński. Learnability Thesis Does Not Entail Church's Thesis. In K. Meer A. Beckmann, E. Csuhaj-Varjú, editor, *Lecture Notes in Computer Science, Volume 8493: Language, Life, Limits*, pages 113–122. Springer International Publishing, 2014.

[15] Walter Dean. Models and Recursivity. Available at `https://pdfs.semanticscholar.org/692a/f690a97981d992983abc92b4cfc4385b31c4.pdf`, 2002.

[16] Hartmut Fitz. Church's Thesis and Physical Computation. In *Church's Thesis After 70 Years*, pages 175–219. Ontos Verlag, Frankfurt, 2006.

[17] Robin Gandy. Church's Thesis and the Principles for Mechanisms. In J. Barwise, H.J. Keisler, and K. Kunen, editors, *The Kleene Symposium*, pages 123–148. North Holland, Amsterdam, 1980.

[18] Michał Tomasz Godziszewski and Joel David Hamkins. Computable Quotient Presentations of Models of Arithmetic and Set Theory. *arXiv e-prints*, page arXiv:1702.08350, 2017.

[19] E. Mark Gold. Limiting Recursion. *Journal of Symbolic Logic*, 30:28–48, 1965.

[20] Reuben Goodstein. On the Restricted Ordinal Theorem. *Journal of Symbolic Logic*, 9:33–41, 1944.

[21] Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme. *Monatshefte für Mathematik und Physik*, 38(1):173–198, 1931.

[22] Volker Halbach and Leon Horsten. Computational Structuralism. *Philosophia Mathematica*, 13(2):174–186, 2005.

[23] Michał Heller and Stanisław Krajewski. *Czy fizyka i matematyka to nauki humanistyczne?* Copernicus Center Press Sp. z o.o., Kraków, 2014.

[24] David Hilbert. Über das Unendliche. *Mathematische Annalen*, 95:161–190, 1926.

[25] Edward Vermilye Huntington. New Sets of Independent Postulates for the Algebra of Logic, with Special Reference to Whitehead and Russell's Principia Mathematica. *Transactions of the American Mathematical Society*, (35), 1933.

[26] Richard Kaye. Tennenbaum's Theorem for Models of Arithmetic. In Juliette Kennedy and Roman Kossak, editors, *Set Theory, Arithmetic, and Foundations of Mathematics Lecture Notes in Logic*, number 36. 2011.

[27] Laurence Kirby and Jeff Paris. Accessible Independence Results for Peano Arithmetic. *Bulletin of the London Mathematical Society*, 14(4):285–293, 1982.

[28] Stephen Kleene. On notation for ordinal numbers. *The Journal of Symbolic Logic*, 3:150–155, 1938.

[29] Andrey Kolmogorov. О понятии алгоритма. *Успехи математических наук*, 8(4):175–176, 1953.

[30] Stanisław Krajewski. Is Church's Thesis Unique? In Piotr Urbańczyk Adam Olszewski, Bartosz Brożek, editor, *Church's Thesis. Logic, Mind and Nature*, pages 113–135. Copernicus Center Press, Kraków, 2014.

[31] Leopold Kronecker. Über den Zahlbegriff. *Journal für die reine und angewandte Mathematik*, 101:337–355, 1887.

[32] Gottfried Wilhelm Leibniz. *Die philosophischen Schriften von Gottfried Wilhelm Leibniz*, volume VII. Weidmannsche Buchhandlung, Berlin, 1890.

[33] Wolfgang Lenzen. *Calculus Universalis. Studien zur Logik von G.W. Leibniz.* Mentis, Paderborn, 2004.

[34] Wolfgang Lenzen. Leibniz's Logic. In Dov M. Gannay and John Woods, editors, *Handbook of the History of Logic*, volume 3, pages 1–83. Elsevier, 2004.

[35] Alex Malpass and Marianna Antonutti Marfori, editors. *The History of Philosophical and Formal Logic. From Aristotle to Tarski.* Bloomsberry Academic, 2017.

[36] Witold Marciszewski and Roman Murawski. *Mechanization of Reasoning in a Historical Perspective.* Rodopi B.V., Amsterdam – Atlanta, 1995.

[37] Yuri Matiyasevich. *Hilbert's Tenth Problem.* MIT Press, Cambridge, Massachusets, 1993.

[38] Marcin Mostowski. On Representing Concepts in Finite Models. *Mathematical Logic Quarterly*, 47:513–523, 2001.

[39] Marcin Mostowski. On Representing Semantics in Finite Models. In G. Kurczewski A. Rojszczak, J. Cachro, editor, *Philosophical Dimensions of Logic and Science*, pages 15–28. Kluwer Academic Publishers, 2003.

[40] Marcin Mostowski. Potential Infinity and the Church's Thesis. *Fundamenta Informaticae*, 81:241–248, 2007.

[41] Marcin Mostowski. On Representability in Finite Arithmetics. In A. Cordon-Franco, A. Fernandez-Margarit, and F. F. Lara-Martin, editors, *JAF26, 26ene Journees sur les Arithmetiques Faibles (26th Weak Arithmetics Days)*, pages 53–64. Fenix Editora, Sevilla, 2008.

[42] Marcin Mostowski and Konrad Zdanowski. FM-representability and Beyond. In *Proceedings of the First International Conference on Computability in Europe: New Computational Paradigms, CiE'05*, pages 358–367. Springer Verlag, Berlin, Heidelberg, 2005.

[43] Jan Mycielski. Analysis without Actual Infinity. *Journal of Symbolic Logic*, 46(3):625–633, 1981.

[44] Emil Leon Post. A Variant of Recursively Unsolvable Problem. *Bulletin of the American Mathematical Society*, 52:264–269, 1946.

[45] Hilary Putnam. Trial and Error Predicates and the Solution to a Problem of Mostowski. *Journal of Symbolic Logic*, 30:49–57, 1965.

[46] Paula Quinon. A Taxonomy of Deviant Encodings. In Dirk Nowotka Florin Manea, Russell G. Miller, editor, *14th Conference on Computability in Europe, CiE 2018, Lecture Notes in Computer Science*, volume 10936 LNCS, pages 338–348. Springer Verlag, Kiel, 2018.

[47] Paula Quinon and Konrad Zdanowski. The Intended Model of Arithmetic. An Argument From Tennenbaum's Theorem. In S. Barry Cooper, Thomas F. Kent, Benedikt Löwe, and Andrea Sorbi, editors, *Computation and Logic in the Real World, CiE 2007, Local Procedings*. Sienna, 2007.

[48] Michael Rescorla. Church's Thesis and the Conceptual Analysis of computability. *Notre Dame Journal of Formal Logic*, 48(2):253–280, 2007.

[49] Michael Rescorla. Copeland and Proudfoot on Computability. *Studies in History and Philosophy of Science Part A*, 43(1):199–202, 2012.

[50] Hartley Rogers, Jr. *Theory of Recursive Functions and Effective Computability*. MIT Press, 1967.

[51] Bertrand Russell. *A Critical Exposition of the Philosophy of Leibniz*. Cambridge University Press, Cambridge, 1990.

[52] Stewart Shapiro. Acceptable Notation. *Notre Dame Journal of Formal Logic*, 23(1):14–20, 01 1982.

[53] Stewart Shapiro. *Foundations without Foundationalism: A Case for Second-Order Logic*, volume 17 of *Oxford Logic Guides*. Clarendon Press, Oxford, 1991.

[54] Stewart Shapiro. *Philosophy of Mathematics: Structure and Ontology*. Oxford University Press, Oxford, 1997.

[55] Wilfried Sieg. Step by Recursive Step: Church's Analysis of Effective Calculability. *Bulletin of Symbolic Logic*, 3:154–180, 1997.

[56] Wilfried Sieg. Gödel on Computability. *Philosophia Mathematica*, 14:189–207, 2006.

[57] Robert Irving Soare. Recursively Enumerable Sets and Degrees. *Bulletin of the American Mathematical Society*, 84(6):1149–1181, 1978.

[58] Robert Irving Soare. *Recursively Enumerable Sets and Degrees. A Study of Computable Functions and Computably Generated Sets.* Springer-Verlag, Berlin, Heidelberg, New York, 1987.

[59] Benedict de Spinoza. *Ethics: Demonstrated in Geometric Order*. Cambridge University Press, 2018.

[60] Stanley Tennenbaum. Non-Archimedean Models for Arithmetic. In *Notices of the American Mathematical Society 6*. 1959.

[61] Boris Trakhtenbrot. Невозможность алгоритма для пробемы разрешимости на конечных классах. *Доклады АН СССР*, 70:569–572, 1950.

[62] Alan Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. 42:230–265, 1936–37.

[63] Michał Wrocławski. Representing Numbers. *Filozofia Nauki*, 26(4):57–73, 2018.

[64] Michał Wrocławski. Representations of Natural Numbers and Computability of Various Functions. In Florin Manea, Barnaby Martin, Daniel Paulusma, and Giuseppe Primiero, editors, *15th Conference on Computability in Europe, CiE 2019, Lecture Notes in Computer Science*. Springer Verlag, 2019. Accepted for publication.

[65] Konrad Zdanowski. On Notation Systems for Natural Numbers and Polynomial Time Computations. Unpublished slides from the conference 'Numbers and Truth', 2012.