Exact representations of and computability on real numbers

Mariëlle Stoelinga
Master's thesis 404
in mathematics and computer science
under supervision of dr. Erik Barendsen

Nijmegen, March 1997

Abstract

This master's thesis explores computability of real functions via representations of real numbers.
We introduce the notion of representation system, which is a set of numerical functions representing all reals. A real function is called computable with respect to some representation system if there exists a corresponding computable functional on that representation system. We investigate two characterizations of computable functionals. We classify the sets of computable real functions of different representation systems, using the notions of "recursive translation" between two systems and "effective approximation by rational numbers."
We also study the set of computable real numbers of a system and compare the set of computable functions on all reals and on the computable reals.
We apply the above results to the well-known representation stems, being Dedekind cuts, B-ary expansions, nested intervals and Cauchy sequences. We determine the computability of $+$, sin and $<$. These systems yield the same set of computable real numbers.

Contents

Chapter 0

Introduction

0.1 About this master's thesis

Although we are not consciously aware of it, real numbers play an important role in daily life in practical as well as in philosophical sense. The real numbers are formed according to the way we experience reality – note the similarity in sound. – The time continuum is described by $\mathbb{R}$, the space around us to be $\mathbb{R}^3$. Classical physics describes reality in terms of real numbers. Apart from time and space, energy, mass, temperature also take values in $\mathbb{R}$. Many physical processes, like magnetic flux and orbits of planets, are described by constructs from real analysis, like differential equations.
Moreover, real numbers are involved in controlling reality. If an airplane takes off, complex calculations are made to overcome gravity. The computer nowadays is indispensable for this.
Therefore it is disturbing to notice that the real numbers have been implemented rather miserably. The REALS, that are representations of real numbers implemented in most computers, contain only representations of finitely many rational numbers. The representation of a real numbers therefore mostly a rounding of it.
The accumulation of relatively small rounding errors may cause an enormous deviation of the exact result. Notably, there exist a sequence that converges to 0, but whose calculation by a computer always diverges, irrespective of the internal precision of the machine. Physicists being cautious with measure inaccuracies, a computer that specifies a result with a precision of 64 decimals, gives only pseudo-security.
The question arises whether or not this situation can be improved. Is a implementation of all real numbers possible? The problem with the representation real numbers is that there are so many of them. It is not possible to have a finite description every one. Therefore it is impossible to represent all of them in a computer, even if we would have an infinite memory at our disposal. However, two solutions are proposed:
Solution one: We do not represent a real number all at once, but specify it stepwise; providing more information about the real in each step. The result of a computation will then also be created in the course of time. Following this

solution, all reals can be represented. A set of objects that represent all real numbers in this way is called a *representation system.*
Solution two: Within the real numbers a subset exist whose elements all have a finite description. The set of computable numbers of a representation system is such a set. The idea is to form the set of computable objects in a representation system. Then a computable real can be described by an algorithm that yields a representation of a real number.
Both solutions do not exclude each other; they can be implemented together within one system. The first one is more general.
A certain data type is implemented to perform operations on. So the following question should be: 'Which functions on real numbers are computable?' In solution one, real numbers are represented by infinite objects. In order to study computable functions on real numbers we need higher order recursion theory, in which we model these functions as functionals.
The computable reals, proposed in solution two, are all finite objects. Computable functions on the computable real numbers can therefore, be modeled as partial recursive functions.
We compare the set of computable real functions induced by both solutions.
Finally we remark that the strategy set out above to study computability on real numbers can be applied also to other uncountable sets. In the practice of computer science such objects are infinite graphs, sets of natural numbers, streams of input in real-time processes, etc.

Research purpose and method

Towards a better implementation of real number arithmetic, this master's thesis studies computability on real numbers. Computations take place on concrete *representations* of real numbers rather than on abstract mathematical objects. By the uncountability of $\mathbb{R}$, it follows that is is impossible to represent all real numbers by a finite description. We explore the following questions:

1. How can the real numbers be represented?
2. Which functions are computable on these representations?
3. Can we indicate a subset of real numbers in which all real numbers have a finite description?
4. How can we characterize the set of computable functions on this set?

Concerning question one, we introduce the notion of a *representation system* of real numbers, being a set in which each real numbers has at least one representation. We study several standard representation systems, known in literature. In essence these are all subsets of $\mathbb{N} \to \mathbb{N}$. We develop a notion of *representation of real function* in terms of a representation system. We would like to call a real function computable if it has a computable representation. Therefore, we develop a notion of *computability* on representation systems. As a representation system consists of functions, we have to study *computable functionals.*

We compare two notions of computability on functionals. The *partial recursive functionals* are an extension of the partial recursive functions. The *effectively continuous functionals* are based on an effective version of continuity in the positive information topology. We focus on functionals that model real functions, which get only total objects as input.
Using the results concerning computable functionals, we explore the computability of three types of operations on representation systems. In particular, we study their computability in the standard representation systems, Dedekind cuts, B-ary expansion, nested intervals and Cauchy sequences. We formulate the *effective approximation property* of a system, stating that, in that system, real numbers can be approximated with rational numbers up to arbitrary precision by a computable functional. Moreover, we consider whether there exist *computable translations*, that map representations in one system to equivalent representations in another system. Finally, we investigate the set of *real functions* that are *computable* with respect to a certain system. We check whether the functions $+$, sin and the predicate < 0 are computable in the standard systems. We also show the relation between the computability of approximations, of translations and of real functions.
Concerning subsets of real numbers that have a finite representation, we introduce a notion of *computable real numbers* with respect to a representation system. We generalize the various definitions from literature, where this set is usually defined in an ad hoc manner on a particular representation system. We compare the sets of computable reals with respect to different systems. As computable real numbers all have a finite description, computable functions on the computable real numbers can be modeled as partial recursive functions, called *effective operations*. Then two notions of computability exist on this set one in terms of functionals and another in term of functions. We also compare these. Finally we prove that computable real numbers with respect to a system can be obtained via application of computable real functions of that system.

Results

We prove three fundamental results on partial recursive functionals: The Normal Form Theorem, continuity in the positive information topology and sequentiality. We prove that the effectively continuous functionals are an extension of the partial recursive ones. As the former need not to be sequential, this is a proper extension. When restricted to total objects, in particular to representations of real numbers, partial recursiveness and effective continuity coincide.
Concerning computability on representations of real numbers, we prove that the standard representation systems satisfy the effective approximation property. The effective approximation property is equivalent to the existence of a translation to interval representations. We show that the notion of computability of real functions depends essentially on the representation. The standard systems yield different classes of computable real functions, which we show by examining the functions $+$, sin and the predicate < 0. If two systems can be effectively translated into another, these yield the same class of computable

functions. Therefore not all standard systems can translated recursively into each other. We also prove that the computable real functions with respect to interval representations are continuous in the Euclidean topology.
Concerning computable real numbers, we show that in general the set of computable real numbers depends on the representation; in the standard systems, these sets coincide and have nice properties. We show that the computable real numbers in a systems can be generated by the computable real functions in the same system.
Every computable functional on the computable reals corresponds to an effective operation. The converse however remains an open problem.

About a system in practical use

We can image a computer implementation based on these ideas working as follows: The implementation of a real function, like sin, gets a representation of a real number as input. This representation is a function, say α, in $\mathbb{N} \to \mathbb{N}$ that is given as a *stream*, that is, by successively specifying $\alpha(0), \alpha(1), \alpha(2), \ldots$. The implementation of sin is lazy and successively produces the output $(\sin \alpha)(0)$, $(\sin \alpha)(1)$, ... It waits for more input until it has got enough information to compute the next element in the output stream. We do not make any assumptions about the amount of input elements needed to produce a certain amount of output: In the case of the sinus we have information about the result even without input as $-1 \leq \sin x \leq 1$ for all $x \in \mathbb{R}$. Using the technique of pipe lining, the stream $\sin f$ can be input to another function.
Functions on the computable reals can be implemented in two ways: Given a Gödel number e of a computable real — a computer program to compute ϕ_e is in fact a Gödel index — the stream $\phi_e(0), \phi_e(1), \ldots$ can be computed and thus be input to the implementation of the sinus described above.
Another way to implement a function on computable numbers is to compute on indices directly. Given e, compute the index of the result. We will prove that any function on computable reals that is computable using streams is also computable on indices; the converse, however, remains an open problem. Given an index of a computable real number, we can generate the associate stream. On the other hand, a computable number can be generated by a computable function on all real numbers.

Related work

A constructive approach to real analysis has been elaborated in various metamathematical settings. Intuitionism, Russian constructivism and classical recursion theory all have worked out their own ideas about the (constructive) nature of real numbers. An overview can be found in [BR87] and in [Beeson85]. We start from a classical definition of the real numbers and study which operations are computable on real numbers, relative to the computability of real numbers. Among all reals, recursive reals have our interest.
Computable reals on higher type have been studied from the 50s by Kleene, introducing schemes S1–S9 [Kleene59] and Platek [Platek66]. Computable func-

tionals, which we use to describe computable real functions, are investigated by Grzegorczyk [Grzegorczyk55]. Nowadays his treatment is still relevant, but may seem somewhat old-fashioned. A more modern approach is followed by Odifreddi [Odifreddi89] en Rogers [Rogers67]. The work of Odifreddi was quite intuitive. However, the results we needed were only hinted at.
The subject of restricted functionals seems not to have been studied extensively before. Functionals restricted to total functions are involved in HEO, which has been studied by Troelstra [Troelstra73].
Grzegorczyk has also applied computable functionals to functions of real numbers, see [Grzegorczyk57]. His work is often taken as a starting point. Most papers — including those of Grzegorczyk himself — choose one particular representation system and elaborate all theory within that system. Applying and refining the theory of computable functionals, we have developed the theory of computability on real numbers in a more general framework, that of a representation system. We have compared several concrete representation systems.
Computability on real numbers fits in the framework of Pour-El and Richards [Pour-El] who axiomatize the notion of a computability structure on a Banach space.
Early investigations in the field of recursive real numbers have been done by Turing [Turing36], Rice [Rice54] and Mazur [Mazur63]. Petèr, [Peter51] Robinson [Robinson52] and Bridges [Bridges94] also have contributed to this research topic. Moschovakis presents an axiomatic characterization of the recursive reals in [Moschovakis65]. Just as with the computability of real numbers, we have developed out notions concerning recursive real numbers within a representation system in general, whereas the literature mostly chooses one system. By studying computability on all real numbers as well as on the computable real numbers, we profit from theory developed to describe computability on all real numbers, when studying the computable real numbers.
Unfortunately, the literature about computable functionals and about computable functionals on real numbers was difficult to access. We have proven many result by ourselves, probably more than necessary.

Future work

This master's thesis has left some open problems that could be solved by future research. We list these here.

- Can every (partial) recursive effective operation be lifted on a representation system be lifted to an effectively continuous functional?
- Is every real function that is computable with respect to $\mathbf{R}_{B\text{-ary}}$ continuous in the Euclidean topology?
- Is it possible to formulate a nice property for a representation system (v, π) to ensure that $\pi(V^c) = \mathcal{R}$?

Beside the concrete representation systems we have treated, other standard representation systems of real numbers, such as continued fraction, can be studied.

Toward a realistic implementation of real numbers in the way we described, a lot of research has to be done. Firstly efficiency has to be taken into account. Some research has been done by [BSS85]. If it comes upto writing programs, is there a or less standard way to implement a real function, for instance from a Taylor series expansion? Can we use approximation techniques from numerical analysis?
The methods used to describe computability on real functions can also be applied to other uncountable sets, like those of infinite graphs, infinite lists, sets of natural numbers, etc.
Therefore we think it is worth developing a more general framework in which computability on all these objects can be described, Starting from a specification, how are the representation systems for these objects related with respect to their computational properties?

0.2 Overview

This master's thesis has been divided into four parts. Chapter one is about real numbers, the topic of chapter two is computability on function spaces, the third chapter treats computability on real numbers and chapter four at last is concerned with computable real numbers and computable functions on these.

Chapter one defines the notion of representation system of real numbers, being a set in which every real number can be represented. Two requirements should exclude representation systems that are not useful in practice. First of all, their elements should be easily implementable in a computer. Therefore, we require representations to be functions from $\mathbb{N}$ to $\mathbb{N}$. Furthermore, it should be possible to approximate a real number effectively up to any given precision from its representation.
Each real function now can be implemented by a function on a representation system that respects the equivalence relation $\equiv$, "represent the same number." This is a function with functions as in- and as output — a functional. Then, computability of real functions can be defined in terms of a computable functionals.
Moreover this chapter treats some standard representation systems. We look at Dedekind cuts, B-ary expansions (if $B = 10$ this yields the common decimal representations for real numbers), nested intervals and Cauchy sequences. Chapter three explores computability on these systems.

Chapter two studies computable functionals. Two notions of computability are defined: partial recursiveness and effective continuity.
A partial recursive functional is a generalization of a partial recursive function. Three fundamental theorems are proven: The Normal Form Theorem, continuity in the positive information topology and sequentiality.
The effectively continuous functionals are an extension of the partial recursive ones.
Functionals that model real functions only get total objects as input. Therefore

we pay special attention to computable functionals – in both senses – with a restricted domain. We show that a computable restricted functional is just the restriction of a computable functional. An important result is that, restricted to total objects, partial recursiveness and effective continuity coincide. Throughout this chapter the positive information topology plays a major role.

Chapter three is about computability on real numbers. Being equipped with a notion of computability on function spaces, we can conclude computability and non-computability of operations rather easily. Three types of functions on representation systems are considered.
Firstly, we prove that, in the standard representation systems, real numbers can be approximated with rational numbers up to arbitrary precision by a computable functional. Moreover, we consider whether there exist computable functionals that translate representations in one system to equivalent representations in another system. Both positive and negative results will be derived for the standard systems. Finally, the notion of computable real function will be defined. The set of computable real functions depends on the representation of real numbers. We check whether the functions $+$, sin and the predicate < 0 are computable in the standard systems. The answers are different in each system. We also study under what conditions the computable real functions are continuous in the Euclidean topology.

Chapter four deals with the set of computable objects within a given representation system. These are the computable real numbers in the system. In general, the set of computable real numbers depends on the representation. However, in the standard systems, these sets coincide and have nice properties.
Computable functions from computable reals to computable reals can be modeled by partial recursive functions on the set of Gödel numbers of the objects. Such functions should respect the relation "represent the same real number" on indices. We call them effective operations. We prove that every computable functional on the computable reals corresponds to an effective operation. However, the converse problem remains open.

0.3 Preparations

Partial functions

A partial function f from a set A to a set B is an object that assign a unique element from B to each element from its domain, which is a subset of A. The space of partial functions from A to B is denoted by $A \rightharpoonup B$.

We call f a function if $\mathrm{Dom}(f) = A$. [1] The set of functions is denoted by $A \to B$.

Let $f, g \in A \rightharpoonup B$ and $x, y \in A$. The domain of the function f is denoted by $\mathrm{Dom}(f)$. If $x \in \mathrm{Dom}(f)$, we say "$f(x)$ is defined" and write $f(x)\downarrow$. If $x \notin \mathrm{Dom}(f)$, then "$f(x)$ is undefined" or $f(x)\uparrow$. Two function applications $f(x)$ and $g(y)$ are equal, notation $f(x) \simeq g(y)$ is they are both undefined or both defined with the same value. Thus $f(x) \simeq f(y) \stackrel{\mathrm{d}}{=} (f(x)\uparrow \,\&\, g(y)\uparrow) \vee (f(x) = g(y))$. The equality of functions is extensional. We have $f = g \stackrel{\mathrm{d}}{=} \mathrm{Dom}(f) = \mathrm{Dom}(g) \,\&\, \forall x \in \mathrm{Dom}(f)[f(x) = g(x)]$. We write $f =_n g$ is f and g agree upto n: $\forall x < n[f(x) = g(x)]$. A function can be restricted to a subset $A' \subseteq A$. We write $f \restriction_{A'}$ to indicate the function in $A \rightharpoonup B$ such that $\forall x \in A'[f \restriction_{A'}(x) \simeq f(x)]$.

If $f : \mathbb{N} \rightharpoonup \mathbb{N}$ then $\mu x[f(x) = 0]$ denotes the smallest $n \in \mathbb{N}$ such that $f(x) = 0$.

A function is **finite** if its domain is finite.

A partial function $f : A \to B$ can be viewed as a set of pairs $\{(x, f(x)) \mid x \in \mathrm{Dom}(f)\}$. Conversely, a set of pairs V can be considered as a function if it is single valued, i.e. $\forall x, y_1, y_2 \in V$: (x, y_1) & $(x, y_2) \implies y_1 = y_2$. In other words, there exists a bijection between the function space $A \to B$ and the collection of single valued sets. We will make use of this correspondence tacitly. In particular we will define functions by naming their associated sets. So, the symbol $\emptyset$ denotes the function that is undefined everywhere on its domain; the set $\{(x, y)\}$ the function that is defined on x only, having value y. If two functions f and g coincide on the intersection of their domains, the expression $\mathbf{f} \cup \mathbf{g}$ makes sense.

Through this isomorphism it is not difficult to see that the function spaces $A_1 \times A_2 \to B$ and $A_1 \to A_2 \to B$ are isomorphic also. Remark that a function in $f : A_1 \times A_2 \to B$ has another domain than its equivalent in $A_1 \to A_2 \to B$.

In particular, these conventions hold for functions whose domain consist of functions, namely functionals.

Sets

Concerning sets, the following symbols are used:

[1] Such a partial function is often called total. We avoid this word here, because it has another meaning in the context of functionals.

$\mathbb{N}$	the set of natural numbers
$\mathbb{N}^*$	the set of natural numbers unequal to 0
$\mathbb{Z}$	the set of integers
$\mathbb{Q}$	the set of rational numbers
$\mathbb{Q}^+$	the set of positive rational numbers
$\mathbb{R}$	the set of real numbers
$\mathbb{C}$	the set of complex numbers

Chapter 1

Representations of real numbers

Real numbers are inspired by our experience of time and space. We have certain ideas about properties of time and space and mathematicians like to derive these formally from a few basic assumptions, called axioms. From the axioms more complex constructions, like sin,lim and $\int$ can be built and general theorems can be proven, like: If $F' = f$ and f is continuous then for all $a, b \in \mathbb{R}$

$$\int_a^b f(x)\mathrm{d}x = F(b) - F(a).$$

On the other hand, if we wish to make statements, like "the length of this path is $\sqrt{2}$" or "the distance light travels through in 1 second, is $2,9979246\ldots\cdot 10^8$ meters", we need to denote, to represent the reals. In the examples above $\sqrt{2}$, 1 and $2,9979246\ldots\cdot 10^8$ are denotations or representations of real numbers. From the axioms of real numbers it follows that it is impossible to give a finite denotation for every real number.

There are several representation systems for real numbers, just like we can denote natural numbers in decimal, binary, etc. notation. But even within one system, there are often several representations for one single real. Compare this to $\mathbb{Q}$, where (in decimal notation) $\frac{1}{2}$,$\frac{2}{4}$ and $\frac{1,000,000}{2,000,000}$ refer to the same element. These have exactly the same mathematical or *extensional* properties. However a computer scientist would say they have different properties. For instance more bits are needed to store $\frac{1,000,000}{2,000,000}$ than to store $\frac{1}{2}$. Intentionally, different representations are distinct objects that satisfy the equivalence relation "represent the same real number."

The mathematical properties of real numbers are determined by their axioms. So the set of real numbers is a structure $< \mathbb{R}, +, \cdot, \leq, 0, 1 >$ that satisfies the axioms. A denotation of a real number refers to a real number, so a representation system of real numbers is a set V together with a function π, mapping a representation to the real number it represents. Calculations are about representations, rather than about abstract properties. A computation takes representations of x and y and constructs their sum, i.e. a representation of $x + y$. Proofs are about abstract properties like $x \leq x + y$.

The following sections work out these ideas more precisely.

1.1 Real numbers and their representations

1.1.1 An axiomatization of real numbers

The properties of real numbers by the second order formula TOFEAS, which stands for Totally Ordered Field, the axiom of Edoxos and Archimedes and the Supremum Axiom. A structure $< \mathbb{R}, +, \cdot, \leq, 0, 1 >$ is a structure of real numbers if:

1. $< \mathbb{R}, +, \cdot, \leq, 0, 1 >$ is a totally ordered field.
2. $\forall x \in \mathbb{R} \forall y \in \mathbb{R} \in \exists n \in \mathbb{N}[x \cdot n > y]$.
3. Every subset of $\mathbb{R}$ that is bounded from above, has a supremum (= lowest upper bound).

From these axioms all functions and definitions from real analysis, like $\sin, \lim, \int$ can be constructed and all theorems from real analysis can be derived; in fact real analysis is all about this. An important property of the formula TOFEAS is that it is categorical, i.e. all structures $< \mathbb{R}, +, \cdot, \leq, 0, 1 >$ that satisfy TOFEAS are isomorphic. From now on, if we speak about $\mathbb{R}$, we mean one of these models. In particular, all models are uncountable, which implies that it is impossible to represent all real numbers by finite objects.

1.1.2 Representations of real numbers

Definition 1.1.1 A **representation system** of real numbers is a tuple (V, π), where $V \subseteq \mathbb{N} \to \mathbb{N}$ and π is a surjective function from V to $\mathbb{R}$. If $\alpha \in V$ and $\pi(\alpha) = x$, then α is said to be a (V, π)**-representation** of x.

We think of elements in V as denotations of real numbers. We have required its elements to be functions. The sole requirement that V is a set would leave open a so many possibilities that comparison, reasoning and defining related notions is not convenient.
Because $\mathbb{R}$ is equinumerous with $\mathbb{N} \to \mathbb{N}$, this is not a serious restriction. Elements in $\mathcal{P}(\mathbb{N})$, $\mathbb{N}^n \to \mathbb{N}^m$, $\mathbb{Q} \to \mathbb{Q}$, sequences of rationals etc. can be encoded as functions in $\mathbb{N} \to \mathbb{N}$.
As we are interested in implementation, we wish the object to be easily implementable. In particular we want to be able to specify a real number as input stepwise. A function f in $\mathbb{N} \to \mathbb{N}$ can be given by an infinite sequence in $f(0), f(1), \ldots$ course of time, called a *stream*.
The function π maps each representation to the real number it refers to. As all numbers should be representable, π should be surjective. It needs not to be injective: A real may have more than one representation. Then π induces an equivalence relation

$$x \equiv_\pi y \iff \pi(x) = \pi(y).$$

We often omit π, if it is clear from V which is meant.

A property we think is crucial in practical use is what we call the effective approximation property. A real numbers is often used to express a quantity, like "the speed of light is $2,9979246\ldots\cdot 10^8$ m/s," as complementary to expressing properties like $x > 0 \ \& \ x^2 = 2$. The numerical value should be derivable from the representation. However, representations are infinite objects, so we do not require that that we can overview the quantity at once — note the dots in the representation of the speed of light. We should be able to approximate the real number $\pi(\alpha)$ with arbitrary precision by rational numbers, of which we believe we can overview the values. Moreover, this approximation should be effective in α. As computability on functions is treated in section 2, the exact definition of the effective approximation property is delayed until 3.1. To avoid going into details of systems that are not suited for implementation, we anticipate on this definition. We only present representation systems that satisfy the effective approximation property, i.e. we slightly adapt the definition of Cauchy sequences.

1.1.3 Representations of real functions

Real functions can be represented in terms of a representation system of real numbers.

Definition 1.1.2 Let (V, π) be a representation system and let $X \subseteq \mathbb{R}$.

1. Let $f : X \to \mathbb{R}$ be a partial function. A function $F : V \to V$ is called a **representation** of f with respect to (V, π), or a (V, π)**-implementation** of f, if for all $\alpha \in \pi^{-1}(X)$

   $$\pi(F(\alpha)) = f(\pi(\alpha)).$$

   In a commuting diagram we have

   $$\begin{array}{ccc} \mathbb{R} & \xrightarrow{f} & \mathbb{R} \\ {\scriptstyle\pi}\uparrow & & \uparrow{\scriptstyle\pi} \\ V & \xrightarrow{F} & V \end{array}$$

2. A (V, π)**-representation** of a function $f : X \to \mathbb{N}$ is a functional $F : V \to \mathbb{N}$ such that for all $\alpha \in \pi^{-1}$ for all $\alpha \in \pi^{-1}(X)$

   $$F(\alpha) = f(\pi(\alpha)).$$

   In a commuting diagram, we have

   $$\begin{array}{ccc} \mathbb{R} & \xrightarrow{f} & \mathbb{N} \\ {\scriptstyle\pi}\uparrow & \nearrow_{F} & \\ V & & \end{array}$$

   The definitions of representations of functions having type $\mathbb{R}^k \to \mathbb{R}$, $\mathbb{N} \to \mathbb{R}$, $\mathbb{R} \to \mathbb{Q}$, etc. are now straightforward.

3. A (V,π)-representation of predicate of a $p \subseteq \mathbb{R}^k$ is a subset $P \subseteq V^k$ such that

$$(\alpha_1 \dots \alpha_k) \in P \iff (\pi(\alpha_1) \dots \pi(\alpha_k)) \in p.$$

Remark 1.1.3 Note that every real function has at least one implementation with respect to some representation system (V,π). If f is a real function, we can define $F : V \to V$ using the Axiom of Choice. If $x \in V$ choose $y \in V$ with $\pi(y) = f(x)$ and take $F(x) = y$.
On the other hand, every function $F : V^n \to V$ that preserves $\equiv_\pi$, can be lifted to a function on $\mathbb{R}$. A function that, like F, is defined on functions, is called a *functional*. As we wish to study computability on representations of real numbers, we should study computable functionals.
In every representation system we have representations of 0 and 1 in V. Also the functions $+$ and $\cdot$ and the relation $\leq$ can be represented. If we examine the set of equivalence classes, $V/\equiv_\pi = \{[x]_{\equiv_\pi} \mid x \in V\}$, we can lift $+, \cdot, \leq$ to V. Then $< V/\equiv_\pi, +, \cdot, \leq, [0]_{\equiv_\pi}, [1]_{\equiv_\pi} >$ is a model of TOFEAS. In fact the traditional models of TOFEAS are constructed this way, i.e. by taking a representation system and then forming the quotient according to $\equiv_\pi$.

1.1.4 Common representations

This section presents some frequently used representations of real numbers and some of their elementary properties, namely Dedekind cuts, B-ary expansions, nested intervals and Cauchy sequences. The representations are all constructed from the rational numbers.
There is a wide range of equivalent forms of every definition that have the same computational properties. We mention some of them, without proof.
As a running example, the number e is represented in all systems. In some representations, the Taylor series expansion is useful. Expansion of the function e^x at 0 yields

$$\left| e - \sum_{k=0}^{n} \frac{1}{k!} \right| \leq \frac{1}{(n+1)!},$$

so the number e equals

$$e = \sum_{k=0}^{\infty} \frac{1}{k!}.$$

We also show a representation of the function $+$ in each system.

Dedekind cuts

Definition 1.1.4 A **Dedekind cut** is a prefix $\alpha \subseteq \mathbb{Q}$,
satisfying

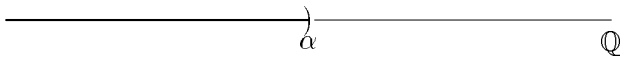
1. $\alpha \neq \emptyset$, $\alpha \neq \mathbb{Q}$,

2. $x \in \alpha, y \leq x \implies y \in \alpha$,
3. $x \in \alpha \implies \exists y \in \alpha[x < y]$ (there is no largest element in α).

The set of Dedekind cuts – or Dedekind representations – is called $\mathbf{R}_{\text{ded}}$.
The function $\pi_{\text{ded}} : \mathbf{R}_{\text{ded}} \to \mathbb{R}$ is defined by

$$\pi_{\text{ded}}(\alpha) = \sup(\alpha).$$

In a picture, we have



As Dedekind cuts are bounded from above, the supremum exists, so π_{ded} is well-defined. Furthermore is surjective. As rational numbers can be encoded by natural numbers, the characteristic function of a subset $\alpha \subseteq \mathbb{Q}$,

$$\chi_\alpha : \mathbb{Q} \to \{0, 1\},$$
$$\chi_\alpha(q) = \begin{cases} 1 & \text{if } q \in \alpha, \\ 0 & \text{otherwise} \end{cases}$$

can be represented by a function of natural numbers. So indeed $(\mathbf{R}_{\text{ded}}, \pi_{\text{ded}})$ is a representation system of real numbers.
Note that π_{ded} is injective, which implies that every real has a unique Dedekind representation and that

$$\alpha(q) = 1 \iff q \in \alpha \iff q < \pi_{\text{ded}}(\alpha).$$

Alternative definitions of Dedekind cuts use suffixes of $\mathbb{Q}$ instead of prefixes or leave out the third requirement above.

Example 1.1.5 The Dedekind representation of e is

$$\eta(q) = \begin{cases} 1 & \text{if } q < e, \\ 0 & \text{if } q > e. \end{cases}$$

Viewed as sets, addition on Dedekind cuts is performed as

$$\alpha \oplus \beta = \{p + r \mid p \in \alpha \;\&\; r \in \beta\}.$$

In terms of characteristic functions, the function $+$ can be represented by

$$\oplus_{\text{ded}} : \mathbf{R}_{\text{ded}} \times \mathbf{R}_{\text{ded}} \to \mathbf{R}_{\text{ded}},$$
$$(\alpha \oplus_{\text{ded}} \beta)(q) = \begin{cases} 1 & \text{if } \exists p, r \in \mathbb{Q}[\alpha(p) = 1 \;\&\; \beta(r) = 1 \;\&\; p + r = q], \\ 0 & \text{otherwise.} \end{cases}$$

Note that we have given $\oplus_{\text{ded}}$ in applicative style.

B-ary expansions

Before introducing the B-ary expansion we need some auxiliary notions:

Definition 1.1.6 A number $B \in \mathbb{N}$, $B \geq 2$ is called a **base** or **radix**.

Definition 1.1.7 Let $B \in \mathbb{N}, B \geq 2$. A number $q \in \mathbb{R}$ is a **B-ary fraction** if it is a fraction having denominator B^n, i.e. there exist $n \in \mathbb{N}$ and $a \in \mathbb{Z}$ such that

$$q = \frac{a}{B^n}.$$

Definition 1.1.8 Let $B \in \mathbb{N}$ be a base. A **B-ary expansion** is a sequence $\alpha_0, \alpha_1, \ldots$ with $\alpha_0 \in \mathbb{Z}, 0 \leq \alpha_n < B$, $n \geq 1$. Here, α_0 should be represented as a B-ary integer. The symbol $\mathbf{R}_{B\text{-ary}}$ denotes the set of B-ary expansions.
The real number represented by $\alpha \in \mathbf{R}_{B\text{-ary}}$ is

$$\pi_{B\text{-ary}}(\alpha) = \sum_{i=0}^{\infty} \frac{\alpha_i}{B^i}.$$

In junction with usual practice we also write $[\alpha]_B$ in stead of $\pi_{B\text{-ary}}(\alpha)$.
Given $\alpha \in \mathbf{R}_{B\text{-ary}}$ and $k \in \mathbb{N}$, $\overline{\alpha}^k$ is the rational that is represented by the first $k+1$ digits of α, so

$$\overline{\alpha}^k = \sum_{i=0}^{k} \frac{\alpha_i}{B^i},$$

then for all k

$$\left| \pi_{B\text{-ary}}(\alpha) - \overline{\alpha}^k \right| \leq \frac{1}{B^k}.$$

If $\alpha \in \mathbf{R}_{B\text{-ary}}$ ends with all zeros, the tail of zeros may be omitted, like in $0.5 \in \mathbf{R}_{B\text{-ary}}$. In that case we speak of a **finite B-ary expansion**. It is clear that $x \in \mathbb{R}$ has a finite B-ary expansion if and only if it is a B-ary fraction. Real numbers, except for B-ary fractions, have a unique B-ary expansion, e.g. $[0.123]_{10} = [0.122999\ldots]_{10}$. If $k = 0$ or $\alpha_k \neq B-1$, then

$$\begin{aligned}
&\pi_{B\text{-ary}}(\alpha_0, \ldots \alpha_k (B-1)(B-1)(B-1)\ldots) = \\
&\overline{\alpha}^k + \sum_{i=k+1}^{\infty} \frac{B-1}{B^i} = \\
&\overline{\alpha}^k + \frac{1}{B^k} = \\
&\pi_{B\text{-ary}}(\alpha_0, \ldots (\alpha_k + 1), 000\ldots)
\end{aligned}$$

Example 1.1.9 For $B = 10$ this yields the familiar decimal expansion of e, which is $2,718128185904\ldots$.

The sum of two B-ary representations is given by

$$\oplus_{B\text{-ary}} : \mathbf{R}_{B\text{-ary}} \to \mathbf{R}_{B\text{-ary}},$$

$$(\alpha \oplus_{B\text{-ary}} \beta)(n) = \begin{cases} \alpha(n) +_B \beta(n) & \text{if } \sum_{k=n+1}^{\infty} \alpha(k) + \beta(k) < B^n, \\ \alpha(n) +_B \beta(n) +_B 1 & \text{otherwise.} \end{cases}$$

$$\text{where } a +_B b = \begin{cases} a + b & \text{if } a + b < B, \\ 0 & \text{otherwise.} \end{cases}$$

This formula says that addition on two B-ary is pointwise addition, passing carries to the left.

Nested intervals

Definition 1.1.10 An **interval representation** is a sequence of tuples $\langle p_0, q_0\rangle$, $\langle p_1, q_1\rangle, \ldots$ such that for all n

1. $p_n, q_n \in \mathbb{Q}$,
2. $p_n \leq q_n$,
3. $\lim_{n\to\infty} q_n - p_n = 0$,
4. $p_n \leq p_{n+1}$ and $q_{n+1} \leq q_n$.

A segment $r = \langle p_n, q_n\rangle$ is associated with the closed interval $[p_n, q_n]$. Its left end p_n is notated by r', the right end by r''. The length of r, $\mathrm{lng}(r)$, is the number $r'' - r'$. The set of interval representations is written as $\mathbf{R}_{\text{int}}$. A sequence of intervals represents the number in its intersection; so take $\pi_{\text{int}} : \mathbf{R}_{\text{int}} \to \mathbb{R}$:

$$\pi_{int}(\alpha) = \lim_{n\to\infty} \alpha'_n,$$

which equals

$$= \lim_{n\to\infty} \alpha''_n,$$

then

$$\alpha \equiv_{\pi_{\text{int}}} \beta \iff \forall n[\alpha''_n \leq \beta'_n \;\&\; \beta''_n \leq \alpha'_n].$$

Other definitions of interval representations make additional demands such as p_n, q_n to be B-ary fractions, or restrict the length of the n^{th} interval, e.g. $q_n - p_n < \frac{1}{2^n}$.

Example 1.1.11 An interval representation α of e is obtained by taking

$$\alpha'(n) = \sum_{k=0}^{n} \frac{1}{k!} - \frac{1}{(n+1)!},$$

$$\alpha''(n) = \sum_{k=0}^{n} \frac{1}{k!} + \frac{1}{(n+1)!}.$$

The function $+$ can be represented as

$$\oplus_{\text{int}} : \mathbf{R}_{\text{int}} \times \mathbf{R}_{\text{int}} \to \mathbf{R}_{\text{int}}$$
$$(\alpha \oplus_{\text{int}} \beta)(n) = <\alpha'(n) + \beta'(n), \alpha''(n) + \beta''(n)>.$$

Cauchy sequences

Definition 1.1.12 A sequence $\alpha_0, \alpha_1, \ldots$ of rational numbers is a **Cauchy sequence** if the differences between the elements become arbitrary small, i.e. if there is a function $c : \mathbb{N} \to \mathbb{N}$ such that for all $k \in \mathbb{N}$

$$\forall n, m \geq c(k) \left[|\alpha_n - \alpha_m| < \frac{1}{2^k} \right].$$

This function c is called a **modulus of convergence**. The set of pairs (α, c) is called $\mathbf{R}_{\text{cauchy}}$. This pair represents the number

$$\pi_{\text{cauchy}}(\alpha, c) = \lim_{n \to \infty} \alpha_n.$$

From analysis we know every Cauchy sequence converges. In chapter 3.1 it will become clear why the modulus is an essential part of a Cauchy representation.

Example 1.1.13 The sequence

$$\alpha(n) = \sum_{k=0}^{n} \frac{1}{k!}$$

converges to e and

$$|e - \alpha(n)| \leq \frac{1}{(n+1)!} \leq \frac{1}{2^n}.$$

So if we define

$$c(n) = n.$$

then the pair (α, c) is a Cauchy representation of e. An implementation of $+$ in Cauchy sequences is

$$\oplus_{\text{cauchy}} : \mathbf{R}_{\text{cauchy}} \times \mathbf{R}_{\text{cauchy}} \to \mathbf{R}_{\text{cauchy}}$$
$$(\langle \alpha, c_1 \rangle \oplus_{\text{cauchy}} \langle \beta, c_2 \rangle)(n) = \langle \alpha(n) + \beta(n), c_1(n) + c_2(n) + 1 \rangle.$$

The list of representations we presented here is not exhaustive. There exist many other representations of real numbers, like: continued fractions, n!-ary expansions, B-ary expansions where negative digits are allowed.

1.1.5 Complex numbers

Another structure analysists are interested in, is that of the complex numbers, $< \mathbb{C}, +, \cdot, 0, 1 >$. Like for the reals, there exists a categorical axiomatization. Common models of these axioms are constructed from models of real numbers, by taking $\mathbb{C} = \mathbb{R}^2$, then representations of complex numbers are pairs of representations of real numbers. From a computational point of view, $\mathbb{R}$ and $\mathbb{R}^2$ do not differ essentially. All theory we build to describe computability on $\mathbb{R}$ can easily be adapted to treat computability on $\mathbb{C}$.

Chapter 2

Computable functionals

Functionals, or higher order functions, are functions which have functions as arguments. Common examples of operations on functions are: integration, differentiation and also primitive recursion and minimalization.

Having functions on real variables in mind, our first interest goes to second order functionals. These are functionals with partial functions on natural numbers as input and as output. Or, equivalently, functionals with partial functions and natural numbers as input and natural numbers as output.

This chapter explores two notions of computability on type level two functionals: partial recursiveness in section 2.1 and effective continuity in 2.2.

The former is a generalization of the notion partial recursive function. Values of the input functions may be used in the computation like those of the initial functions.

We prove three fundamental properties of partial recursive functionals. The Normal Form Theorem (2.1.10) is a generalization of the Normal Forms Theorem in recursion theory. An important consequence of this theorem is the monotony and compactness of partial recursive functionals (theorem 2.1.14). The third theorem (2.1.18) also follows by the normal form theorem. It states that partial recursive functionals are sequential, which means that there exist a sequential algorithm to evaluate a partial recursive functional on its input. This implies that a partial recursive functional is determined by its behaviour on finite functions.

The second notion of computable functional, defined in 2.2, is called effective continuity, because such a functional is continuous in the positive information topology (2.2.1). Continuity in this topology is equivalent to monotony and compactness (theorem 2.2.3). We show in 2.2.12 that effective continuity is a proper extension of partial recursiveness.

Section 2.4 prepares for the application of functionals in the setting of real numbers. Functionals that represent real numbers only get total functions as input. We consider functionals that are restricted to a certain subset P. It defines equivalents of the both notions of computability for restricted functionals. Subsection 2.4.3 studies computable functionals restricted to a set of total functions. In this case, partial recursiveness and effective continuity coincide.

Preparations

The following notations are used from now on:

$\mathbf{P}_k$	the set of partial functions from $\mathbb{N}^k$ to $\mathbb{N}$, $\mathbf{P} = \mathbf{P}_1$ and $\mathbf{P}_0 = \mathbb{N}$
PR	the set of partial recursive functions from $\mathbb{N}$ to $\mathbb{N}$
R	the set of recursive functions from $\mathbb{N}$ to $\mathbb{N}$
T	the set of (total) functions from $\mathbb{N}$ to $\mathbb{N}$

Concerning variables the conventions are:
F,G, ... range over functionals,
f,g, ... are used for elements of $\mathbf{P}$,
$i, j, k, l, m, n, x, y, \ldots$ are natural numbers.
A vector of elements $(a_1, \ldots, a_n)$ is denoted by $\vec{a}$.

Section 0.3 has pointed out that the structures $\mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_m} \to \mathbf{P}_n$ and $\mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_m} \times \mathbb{N}^n \rightharpoonup \mathbb{N}$ are isomorphic as sets. For sake for simplicity, we identify them. If $F : \mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_m} \to \mathbf{P}_n$, we say that F is in so-called Curry style; $F : \mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_m} \to \mathbb{N}^n \rightharpoonup \mathbb{N}$ is in applicative style.
Functionals are usually treated in applicative style. The result of function application then is a natural number, which is a "visible" object. We prefer to view functionals mapping functions to functions for representations of functions on real numbers map functions, c.q. representations of real numbers, to functions. By taking $n = 0$ we get functionals that map functions to natural numbers, for by definition $\mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_m} \to \mathbf{P}_0 = \mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_m} \to \mathbb{N}$. In particular representations of functions in $\mathbb{R} \to \mathbb{N}$ fit in this framework.
We have remarked in section 0.3 that the applicative and the Curry versions do have different domains. In both the cases above, the set $\mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_m}$, which strictly spoken is the domain of the curried functional, is called the **function domain** of F. It is denoted by $\mathrm{Fdom}(F)$.
If $\vec{f} \in \mathrm{Fdom}(F)$, then $F(\vec{f})$ is a partial function. In particular, the object $F(\vec{f})$ always exists. On the contrary, $F(\vec{f})(\vec{x})$ may be undefined. It is obvious that if $F(\vec{f})(\vec{x})\!\downarrow$ then $F(\vec{f})(\vec{x}) \in \mathbb{N}$. The set $\{(\vec{f}, \vec{x}) \in \mathrm{Fdom}(F) \times \mathbb{N}^k \mid F(\vec{f})(\vec{x})\!\downarrow\}$ is called the **domain** of F. It is the domain of the applicative version.
As functionals are a special kind of functions, everything that is defined in 0.3 is also applicable to functionals.

Definition 2.0.14 A functional F is called **total** if it maps all total functions from $\mathrm{Fdom}(F)$ to total functions.

We use the convention that a functional is strict in its number arguments, $H(\vec{f})(G_1(\vec{f})(\vec{x}), \ldots, G_m(\vec{f})(\vec{x}))$ is undefined whenever $G_1(\vec{f})(\vec{x})$ is undefined for some i.

2.1 Partial recursive functionals

In order to give a definition of partial recursive functionals, we wish to develop an intuitive idea of computability on functions: Given a partial function f and

a number n, which actions on f and n would we call computable? In order to answer this question we first consider what information we may assume to be available of a function if it is input to a functional. Then we investigate how this information can be used in a computation.
The essential property of a function is that it assigns a unique output to each input in its domain. A function *is* its input-output behaviour. The usage of f is independent from its generation, because f is generated outside the computation, for instance by following a function presciption or by throwing a dice. Besides, we do not need to calculate f ourselves and therefore there is no reason to require f to be computable; f may be *any* partial function.
Now we have an idea how f can be used in a computation: Just like n and the result of function application (like $\mathrm{S}(n)$), values of f can be input for further computation. As a calculation is performed step by step a terminating computation will use only finitely many values of f. Theorem 2.1.14 shows our notion of computability indeed has this property.
These considerations lead to the idea to use relative recursiveness to define the partial recursive functionals, i.e. recursiveness with the input functions as extra initial functions. An immediate translation into a definition is: [Odifreddi89]

> A functional $F(f_1, \ldots, f_n, x_1, \ldots, x_m)$ is partial recursive if it can be obtained from $f_1, \ldots, f_n$ and the initial functions S, Z, P_i^n by composition, primitive recursion and minimalization.

The following definition provides a more explicit formulation. It is a modification of the definition in [Grzegorczyk55].

Definition 2.1.1 (partial recursive functionals) For all $m, k_1, \ldots, k_m \in \mathbb{N}^*$ the partial recursive functionals with function domain $\mathbf{P}_{k_1} \times \ldots \times \mathbf{P}_{k_m}$ are inductively defined by:

1. For all $n \in \mathbb{N}$, the followoing initial functionals are partial recursive:

   - the zero functional

     $$\mathrm{Z} : \mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_m} \to \mathbf{P}$$
     $$\mathrm{Z}\,(\vec{f})(x) \simeq 0,$$

   - the successor functional

     $$\mathrm{S} : \mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_m} \to \mathbf{P}$$
     $$\mathrm{S}\,(\vec{f})(x) \simeq x + 1,$$

   - the projection functionals: For all $i, i \leq n$

     $$\mathrm{P}^i_{m,n} : \mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_m} \to \mathbf{P}_n$$
     $$\mathrm{P}^i_{m,n}(\vec{f})(\vec{x}) \simeq x_i,$$

- the application functionals: For all i with $k_i = n$

$$\mathrm{A}^i_{m,n} : \mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_m} \to \mathbf{P}_n$$
$$\mathrm{A}^i_{m,n}(\vec{f})(\vec{x}) \simeq f_i(\vec{x}).$$

2. The following operations build partial recursive functionals from partial recursive functionals:

   - If $G_1, \cdots G_m$ and H are partial recursive functionals,

$$G_i : \mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_m} \to \mathbf{P}_n \text{ and}$$
$$H : \mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_m} \times \mathbb{N}^m \to \mathbb{N}$$

     then

$$\mathrm{Comp}[H, G_1, \ldots G_m] : \mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_m} \to \mathbf{P}_n$$
$$\mathrm{Comp}[H, G_1, \ldots G_m](\vec{f})(\vec{x}) \simeq H(\vec{f})(G_1(\vec{f})(\vec{x}), \ldots, G_m(\vec{f})(\vec{x}))$$

     is a partial recursive functional.

   - If G and H are partial recursive functionals,

$$G : \mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_m} \to \mathbf{P}_n$$
$$H : \mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_m} \to \mathbf{P}_{n+1}$$

     then $\mathrm{Primrec}[G, H]$ is partial recursive. Let, for the moment, $F = \mathrm{Primrec}[G, H]$, then

$$F(\vec{f})(\vec{x}, 0) \simeq G(\vec{f})(\vec{x})$$
$$F(\vec{f})(\vec{x}, y+1) \simeq H(\vec{f})(F(\vec{f})(\vec{x}, y), \vec{x}, y).$$

   - If the functional

$$G : \mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_m} \to \mathbf{P}_{n+1}$$

     is partial recursive, then also is:

$$Min[G] : \mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_m} \to \mathbf{P}_n$$
$$Min[G](\vec{f})(\vec{x}) \simeq$$
$$\begin{cases} \mu y[G(\vec{f})(\vec{x}, y) = 0] & \text{if } \exists y[G(\vec{f})(\vec{x}, y) = 0 \ \& \\ & \quad \forall i < y[G(\vec{f})(\vec{x}, i) \downarrow \ \& \ G(\vec{f})(\vec{x}, i) \neq 0]] \\ \uparrow & \text{otherwise.} \end{cases}$$

3. The rules mentioned under 1. and 2. yield all partial recursive functionals.

Definition 2.1.2 A functional F is **recursive** if it is partial recursive and total.

Recall that totality means that F maps total functions to total functions.

Example 2.1.3 A well-known operation on functions is iteration. The n^{th} iteration of f, notation $f^{(n)}$, is defined as:

$$f^{(n)}(x) = \underbrace{f(f(\ldots f}_{n \text{ times}}(x)\ldots)).$$

Iteration is recursive in f, for

$$f^{(n)}(x) = \text{Primrec}[\text{Comp}[f, \text{P}^1_3], \text{P}^2_2](n, x)$$

As a functional, iteration of f equals:

$$= \text{Primrec}[\text{Comp}[\text{A}^1_{1,3}, \text{P}^1_{1,3}], \text{P}^2_2](f)(n, x).$$

Addition of functions is recursive:

$$Add : \mathbf{P}^2 \to \mathbf{P}$$
$$Add(f, g)(n) = f(n) + g(n).$$

It is easy to see that this functional is partial recursive using the characterization based on relative recursiveness. An exact proof is not difficult using proposition 2.1.5.

The following four propositions state some elementary properties of partial recursive functionals:

Proposition 2.1.4 *[[Odifreddi89], ch II.3] For all partial recursive functionals F and all $\vec{f} \in \text{Fdom}(F)$:*

$$f_1, \ldots f_n \in \mathbf{PR} \implies F(f_1, \ldots, f_n) \in \mathbf{PR}.$$

PROOF: An easy induction on the generation of F. □

Proposition 2.1.5 *[Grzegorczyk55] For all n and all partial recursive functions $\psi : \mathbb{N}^n \to \mathbb{N}$, for all $m, k_1, \ldots, k_m$, there exists a partial recursive functional F such that:*

$$F : \mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_m} \times \mathbb{N}^n \to \mathbb{N},$$
$$\forall \vec{f} \in \text{Fdom}(F) \left[F(\vec{f}) = \psi \right].$$

PROOF: By induction on the generation of ψ. □

Later, in 2.1.2, we will see that is equivalent to the existence of a recursive F with $F(\emptyset) = \psi$. The proof of the latter formulation however is not easier.

Proposition 2.1.6 (substitution property) *[[Odifreddi89], ch II.3] For all m,n,p,$k_1, \ldots, k_m$,$l_1, \ldots, l_p$, if $G_1, \ldots, G_p$ and H are partial recursive functionals, and*

$$H : \mathbf{P}_{l_1} \times \cdots \times \mathbf{P}_{l_m} \times \mathbb{N}^n \to \mathbb{N},$$
$$G_i : \mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_p} \times \mathbb{N}^{l_i} \to \mathbb{N},$$

then also

$$F : \mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_p} \times \mathbb{N}^n \to \mathbb{N}$$
$$F(\vec{f})(\vec{x}) = H(G_1(\vec{f}), \ldots G_p(\vec{f}))(\vec{x})$$

is a partial recursive functional.

PROOF: By induction on the generation of H. □

Proposition 2.1.7 *The partial recursive functionals are closed under bounded minimalization, definition by cases, iteration and course-of-value recursion. This means the following:*

1. *If G is a partial recursive functional of appropriate type, then so is the functional defined by:*

$$Bmin[G](\vec{f})(\vec{x}) \simeq \begin{cases} Min[G](\vec{f})(\vec{x}, y) & \text{if } \mu[G](\vec{f})(\vec{x}) < y \\ y & \text{otherwise} \end{cases}$$

   Here $Min[G]$ is as in definition 2.1.1.

2. *Let $n \in \mathbb{N}$. Suppose $G_1, \ldots G_n, F_1, \ldots F_n$ are partial recursive functionals of appropriate type. Assume $G_1, \ldots G_n$ have pairwise disjoint domains. We can see $\mathrm{Dom}(G_i)$ as a recursively enumerable predicate on functions. Then the functional*

$$Cases(\vec{f})(\vec{x}) \simeq \begin{cases} F_1(\vec{f})(\vec{x}) & (\vec{f}, \vec{x}) \in \mathrm{Dom}(G_1), \\ F_2(\vec{f})(\vec{x}) & (\vec{f}, \vec{x}) \in \mathrm{Dom}(G_2), \\ \vdots & \vdots \\ \uparrow & \text{otherwise}, \end{cases}$$

   is partial recursive.

3. *If $F : \mathbf{P} \to \mathbf{P}$ is a partial recursive functional, its iteration, given by*

$$It[F](f)(x, 0) \simeq f(x)$$
$$It[F](f)(x, k+1) \simeq F(It[F](f))(x),$$
$$\simeq F(F^k(f))(x)$$

   is again a partial recursive functional.

4. *Suppose H is a partial recursive functional. The functional defined by*

$$F(\vec{f})(\vec{x}, y) \simeq H(\vec{f})(\vec{x}, y, \tilde{F}(\vec{f})(\vec{x}, y))$$
where
$$\tilde{F}(\vec{f})(\vec{x}, 0) \simeq 0$$
$$\tilde{F}(\vec{f})(\vec{x}, y+1) \simeq <F(\vec{f})(\vec{x}, 0), \ldots, F(\vec{f})(\vec{x}, y)>$$

   is partial recursive as well.

PROOF: The partial recursive *functions* are closed under bounded minimalization, definition by cases, iteration and course-of-value recursion. It is not difficult to show that these constructions are recursive in their function arguments. By the Substitution Property it follows that the partial recursive functionals are closed under the mentioned operations. We work out the proof for iteration, leaving the other constructions to the reader.
Example 2.1.3 has shown that iteration of a function is partial recursive. Thus,

$$It : \mathbf{P} \to \mathbf{P}_2,$$
$$It(g)(n, x) = g^n(x)$$

is a partial recursive functional. Now by the Substitution Property it follows that $It[F]$ is partial recursive, if F is so.
□

2.1.1 Normal Form Theorem

In order to be able to reason about computations of partial recursive functionals a Normal Form Theorem, like for partial recursive functions, is needed. This theorem, which gives a standard way of computing a functional, is based on the so-called T-predicate. This predicate verifies whether a computation, encoded as a natural number, is a correct computation of the functional under consideration.
Unfortunately, the situation for functionals is not as smooth as for functions. Undefined applications of the input functions may cause the verification process to get stuck. Therefore we do not require the T-predicate to be recursive: Its characteristic functional may be undefined for certain values. In 2.1.15 we prove no recursive predicate can serve as a T- predicate.
In sections 2.1.2 and 2.1.3 we prove two fundamental properties of partial recursive functionals by means of the Normal Form Theorem.
Following the same method as for functions, a T-predicate for partial recursive functionals is defined:

1. Assign code numbers to partial recursive functionals, from which the recursive structure of a functional can be deduced.
2. Introduce computation trees of partial recursive functionals.
3. Encode the computation tree as a natural number.
4. Define a predicate T which tells whether a number z is an encoded computation tree of the functional having code number e on input $\vec{f}$ and $\vec{x}$.

Encoding sequences of natural numbers

There is an injective function $<> : \bigcup_{k\geq 0} \mathbb{N}^k \to \mathbb{N}$, such that the following functions are computable:

- For all k : $<> \restriction_{\mathbb{N}^k} : \mathbb{N}^k \to \mathbb{N}$, the restriction from $<>$ to $\mathbb{N}^k$

- concatenation $\star : \mathbb{N}^2 \to \mathbb{N}$, such that:

$$<x_1, \ldots, x_n> \star <y_1, \ldots, y_m> = <x_1, \ldots, x_n, y_1 \ldots, y_m>,$$

- the length function, $\mathrm{lng} : \mathbb{N} \to \mathbb{N}$, such that:

$$\mathrm{lng}(<x_1, \ldots, x_n>) = n,$$

- selection $() : \mathbb{N}^2 \to \mathbb{N}$, such that:

$$(<x_1 \ldots, x_n>)_k = \begin{cases} x_k & \text{if } 1 \leq k \leq n, \\ 0 & \text{otherwise,} \end{cases}$$

  Parentheses are omitted in successive selections: $\Big(((x)_i)_j\Big)_k$ is written as $(x)_{i,j,k}$

- subsequence $()_: : \mathbb{N}^3 \to \mathbb{N}$, such that:

$$(<x_1 \ldots, x_n>)_{k:l} = \begin{cases} <x_k, \ldots, x_l> & \text{if } 1 \leq k \leq l \leq n, \\ 0 & \text{otherwise,} \end{cases}$$

For the function $<>$ we can take:

$$<> = 0,$$
$$<x_1, \ldots, x_n> = p_1^{x_1+1} \cdot p_2^{x_2+1} \cdots p_n^{x_n+1},$$
where p_k is the k^{th} prime number.

Index numbers of partial recursive functionals

Definition 2.1.8 The set I of **index numbers** for partial recursive functionals is defined inductively:

1. The set I contains indices of the initial partial recursive functionals: For all $m, k_1, \ldots, k_m, n$:
   - the index of $\mathrm{Z} : <0, <k_1, \ldots, k_m>, n> \in I$
   - the index of $\mathrm{S} : <1, <k_1, \ldots, k_m>, 1> \in I$
   - the index of $\mathrm{P}^i_{m,n} : <2, <k_1, \ldots, k_m>, n, i> \in I$, if $i \leq n$
   - the index of $\mathrm{A}^i_{m,n} : <3, <k_1, \ldots, k_m>, n, i> \in I$, if $i \leq m \;\&\; k_i = n$
2. Index numbers for functionals that are constructed from other functionals:
   - by composition: For all $d, n, (e)_1, \ldots, (e)_n \in I$ with $(e_1)_2 = \ldots = (e_n)_2$ and $(e_1)_3 = \ldots = (e_n)_3 = (d)_2$ and $(d)_3 = n$:

$$<4, <k_1, \ldots, k_m>, n, (e_1)_2, d, (e)_1, \ldots, (e)_n> \in I.$$

- by primitive recursion: For all $d, e \in I$ with $(e)_3 = (d)_3 + 2$ and $(e)_2 = (d)_2$:

$$<5, <k_1, \ldots, k_m>, n, (e)_2, (e)_3 + 1, d, e> \in I.$$

- by minimalization: For all $e \in I$ with $(e)_2 \geq 2$:

$$<6, <k_1, \ldots, k_m>, n, (e)_2, (e)_3 - 1, e> \in I.$$

3. The rules mentioned under 1. and 2. yield all elements in I.

Given an index number $e \in I$, $\Phi_\mathbf{e}$ denotes the functional with index number e. From the definition above it is easy to construct this functional.

The index numbers are constructed in such a way that I is a recursive subset of $\mathbb{N}$ and such that:

$(e)_1$	: identification number,
$(e)_2$	$= <(e)_{2,1}, \ldots, (e)_{2,m}>$ arities of input functions,
$(e)_3$	: number of numerical inputs,
$(e)_{i \geq 4}$	: if $(e)_1 = 4, 5, 6$: code numbers of functionals from which Φ_e is constructed, if $(e)_1 = 2, 3$: or additional information.

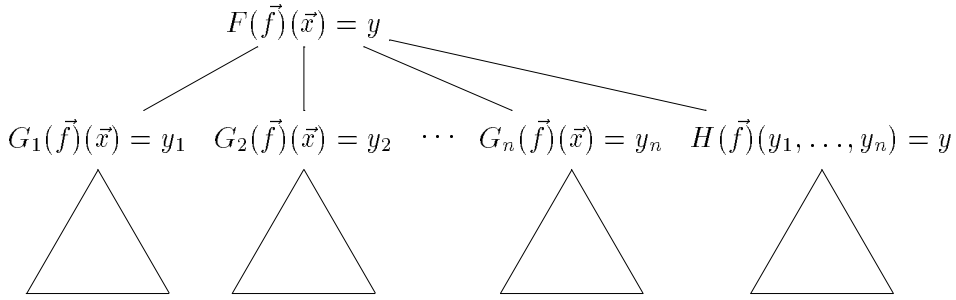
Computation trees

The computation tree of a functional F on function input $\vec{f}$ and numerical input $\vec{x}$ can be represented in canonical form by a computation tree. Each node in the tree corresponds to a step in the computation and sub-results are yielded by subtrees. The value $F(\vec{f})(\vec{x})$ is in the root.
As computation trees are used only to make the definition of the T-predicate more clear, we do not give a very formal definition, it easily follows from the sketch below. If the expression $F(\vec{f})(\vec{x}) = y$ appears in a note, we intend to store the objects $F, \vec{f}, \vec{x}, y$ where $F(\vec{f})(\vec{x}) = y$.
In the formulas below we assume the function and number input to be of appropriate type.

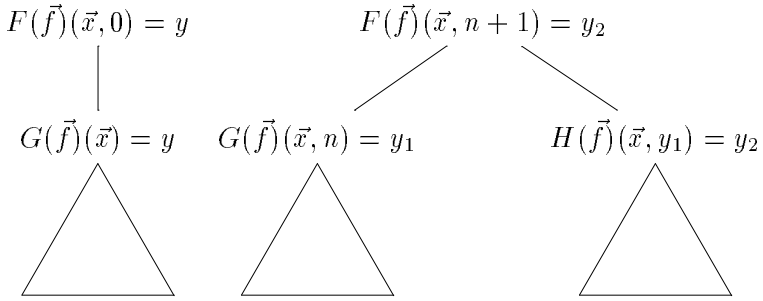
- The computation tree of an initial functional consists of one node.

$$\mathrm{Z}(\vec{f})(x) = 0$$
$$\mathrm{S}(\vec{f})(x) = y$$
$$\mathrm{P}^i_{mn}(\vec{f})(\vec{x}) = x_i$$
$$\mathrm{A}^i_{mn}(\vec{f})(\vec{x}) = f_i(\vec{x})$$

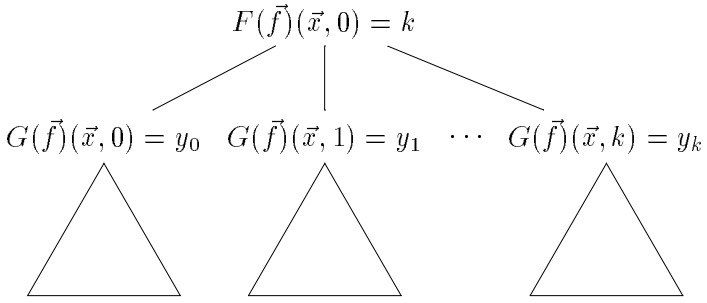
- The computation tree of the functional $F = \mathrm{Comp}[H, G_1, \ldots G_m]$ is built from the computation trees of $G_1, \ldots G_m$ and H.



- The construction of the computation tree of $F = \text{Primrec}[G, H]$ depends on its numerical input.



- The computation the of the functional $\mu[G]$ is constructed from one or more computation trees of G.



where $y_0, \ldots y_{k-1} \neq 0$

We can encode the tree by a natural numbers if

- we represent the functionals by index numbers.
- we do not encode the input functions, which are infinite objects, in the tree but their values may appear in application nodes.

The T-predicate

The T-predicate tells, given $\vec{f}, e \in I, xs = <x_1, \ldots, x_n>$ and z, whether z encodes a computation tree of $\Phi_e(\vec{f})(x_1, \ldots, x_n)$.

For all $m, k_1, \ldots, k_m$ a predicate $T = T_{k_1,\ldots,k_m} \subseteq \mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_m} \times \mathbb{N}^3$ is defined by:

$$
\begin{aligned}
T(\vec{f})(e, xs, z) \iff\ & e \in I \ \&\ (e)_2 = k \ \&\ (e)_3 = \mathrm{lng}(xs) \ \& \\
((e)_1 = 0 \implies\ & z = <(e)_1, xs, 0>) \ \& \\
((e)_1 = 1 \implies\ & z = <e, xs, (xs)_1 + 1> \ \&\ \mathrm{lng}(xs) = 1) \ \& \\
((e)_1 = 2 \implies\ & z = <e, xs, (xs)_{(e)_4}>) \ \& \\
((e)_1 = 3 \implies\ & z = <e, xs, f_{(e)_4}((xs)_1 \ldots (xs)_{(e)_4})> \ \& \\
& (e)_3 = k_{(e)_4} = \mathrm{lng}(xs)) \ \& \\
((e)_1 = 4 \implies\ & \forall i, 4 \le i < \mathrm{lng}(z)[T(\vec{f})((e)_i, xs, z_i)] \ \& \\
& T(\vec{f})((e)_{\mathrm{lng}(e)}, ys, z_{\mathrm{lng}(z)}) \\
& \quad \text{where } ys = <(z)_{4,3}, (z)_{5,3} \ldots (z)_{\mathrm{lng}(z)-1,3}> \ \& \\
& (z)_3 = z_{3,5}) \ \& \\
((e)_1 = 5 \implies\ & \mathrm{lng}(xs) = 0 \implies T(\vec{f})((e)_4, xs_{1:\mathrm{lng}(xs)-1}, z_4) \ \& \\
& \mathrm{lng}(xs) \neq 0 \implies \\
& \quad T(\vec{f})((e)_4, xs_{1:\mathrm{lng}(xs)-1} \star <(xs)_{\mathrm{lng}(xs)} - 1>, z_4)) \ \& \\
((e)_1 = 6 \implies\ & \forall i, 4 \le i \le \mathrm{lng}(z)[T(\vec{f})((e)_4, xs \star <i>, z_i)] \ \& \\
& \forall i, 4 \le i \le \mathrm{lng}(z)[z_{i,3} \neq 0] \ \&\ z_{\mathrm{lng}(z)} = 0 \ \& \\
& (z)_3 = \mathrm{lng}(z) - 3]).
\end{aligned}
$$

We define a partial recursive functional K_T that verifies whether $T(\vec{f})(e, xs, z)$. We stress that K_T is not total, because undefined values of the input functions may cause the verification process to get stuck. To prevent needless nontermination, the case $(e)_1 = 3$ (application of an input function), is performed very carefully: We actually compute as few applications as possible. Given $e, xs, z \in \mathbb{N}$, firstly the syntax of a candidate application in z is examined. We look if z applies the function specified by e to the specified arguments correctly. If indeed we have to do with an application of the right form, the input function is evaluated to see if it produces the output specified by z.

The other cases, $(e)_1 \in \{0, 1, 2, 4, 5, 6\}$ are treated in a straightforward way. We work out the case $(e)_1 = 1$, the others are tedious, but easy.

Let $m, k_1, \ldots, k_m \in \mathbb{N}$. The functional $K_T : \mathbf{P}_{k_1} \times \cdots \mathbf{P}_{k_m}$ is obtained from the T-predicate as follows. Let $k = <k_1, \ldots, k_m>$.

We focus on the case $(e)_1 = 0$:

$$K_T(\vec{f})(e,xs,z) =$$
$$\begin{cases} 1 & \text{if } (e)_1 = 0 \ \& \ z = <e, xs, (xs)_1 + 1> \ \& \ \mathrm{lng}(xs) = 1, \\ 0 & \text{if } (e)_1 = 0 \ \& \ not : (z = <e, xs, (xs)_1 + 1> \ \& \ \mathrm{lng}(xs) = 1), \\ \vdots & \end{cases}$$

We also treat the case $(e)_1 = 3$:

$$K_T(\vec{f})(e,xs,z) =$$
$$\begin{cases} \vdots & \\ 1 & \text{if } (e)_1 = 3 \ \& \\ & \quad \underline{\text{if}} \ (e)_3 = k_{(e)_4} = \mathrm{lng}(xs) \ \& \ (e)_1 = (z)_1 \ \& \ (z)_2 = xs \\ & \quad \underline{\text{then}} \ (z)_4 = f_{(e)_4}((xs)_1, \ldots (xs)_{\mathrm{lng}\, xs}), \\ 0 & \text{if } (e)_1 = 3 \ \& \ not : ((e)_3 = k_{(e)_4} = \mathrm{lng}(xs) \ \& \ (e)_1 = (z)_1 \ \& \ (z)_2 = xs) \ , \\ \vdots & \end{cases}$$

Here the "<u>if</u> ... <u>then</u> ... " is lazy: If the condition is not fulfilled, the <u>then</u>-part will not be evaluated. By proposition 2.1.7 this construction is partial recursive.

The following theorem prepares for the Normal Form Theorem.

Theorem 2.1.9 *Let $m, k_1, \ldots, k_m \in \mathbb{N}$. Let $\vec{f} \in \mathbf{P} \times \cdots \mathbf{P}_{k_m}$ and $e, xs, z \in \mathbb{N}$. Then for $K_T = K_{T_{<k_1,\ldots,k_m>}}$. we have*

1. *K_T is partial recursive.*
2. *$K_T(\vec{f})(e,xs,z) = 1 \iff T(\vec{f})(e,xs,z)$.*
3. *$(K_T(\vec{f})(e,xs,z) = 0 \vee K_T(\vec{f})(e,xs,z)\!\uparrow) \iff not : T(\vec{f})(e,xs,z)$.*
4. *$\exists z[T(\vec{f})(e,xs,z)] \implies K_T(\vec{f})(e,xs,z)\!\uparrow$.*

PROOF:

1. The partial recursive functionals are closed under definition by cases.
2. By induction on e.
3. Idem.
4. By induction on e, essential is our treatment of the case $(e)_1 = 3$.

□

Now we formulate the Normal Form Theorem.

Theorem 2.1.10 (Normal Form Theorem) *Let $m, k_1, \ldots, k_m, n \in \mathbb{N}$. There exists a partial recursive functional $K_T = K_{T_{<k_1,\ldots,k_m>}} : \mathbf{P} \times \cdots \mathbf{P}_{k_m} \times \mathbb{N} \to \mathbb{N}$ and a partial recursive function $U : \mathbb{N} \to \mathbb{N}$ such that for all $e, x_1, \ldots x_n, z \in \mathbb{N}$*

1. $\Phi_e(\vec{f})(<x_1,\ldots x_n>,z)\downarrow \iff \exists z[K_T(\vec{f})(e,<x_1,\ldots x_n>,z)=0]$.
2. $\Phi_e(\vec{f})(<x_1,\ldots x_n>,z)\simeq U(\mu z[K_T(\vec{f})(e,<x_1,\ldots x_n>,z)=0])$.

PROOF: Both parts are proven by induction on e. Define $U:\mathbb{N}\to\mathbb{N}$ by $U(z)=(z)_4$. $\square$

Consequences of the Normal Form Theorem

We show some consequences of the Normal Form Theorem. For sake of simplicity, these are formulated for functionals in $\mathbf{P}\to\mathbf{P}$ only. Section 2.3 indicates how to generalize the results.

2.1.2 Monotony and compactness

Definition 2.1.11 [Odifreddi89] The set $\mathbf{P}$ can be furnished with a **partial order** $\subseteq$. We call f a **subfunction** of g, or g an **extension** of f, notation $f\subseteq g$, if

$$\mathrm{Dom}(f)\subseteq\mathrm{Dom}(g)\ \&\ \forall x\in\mathrm{Dom}(f)\,[f(x)=g(x)]\,.$$

The functional $F:\mathbf{P}\to\mathbf{P}$ is **monotone** if for all $f,g\in\mathbf{P}$:

$$f\subseteq g\implies F(f)\subseteq F(g),$$

F is called **compact** if for all $x\in\mathbb{N}, f\in\mathbf{P}$:

$$\text{if } F(f)(x)\simeq z \text{ then } \exists u\in\mathbf{P}\,[u \text{ finite } \ \&\ F(u)(x)\simeq z]\,.$$

In section 2.2.1 we will give a topological description of the notion compact and monotone.

Proposition 2.1.12 *[Odifreddi89] The condition that F is monotone and compact can be summarized as: For all $f\in\mathbf{P}$ and $x,z\in\mathbb{N}$*

$$F(f)(x)\simeq z\iff\exists u,u\ \textit{finite}[u\subseteq f\ \&\ F(u)(x)\simeq z.]$$

Some examples of monotone and compact functionals are:

Example 2.1.13

$$F(f)(x)=f(f(x+1)),$$
$$G(f)(x)=\begin{cases}0 & \text{if } f(4)\downarrow\vee f(5)\downarrow,\\ \uparrow & \text{otherwise,}\end{cases}$$
$$\text{for } G(f)(x)\simeq G(u)(x),\ \text{where } u=f\restriction_{\{4,5\}}.$$

Theorem 2.1.14 *[Odifreddi89] The partial recursive functionals are monotone and compact.*

PROOF: Let $F : \mathbf{P} \to \mathbf{P}$ be a partial recursive functional.
monotony: Let $f, g \in \mathbf{P}$. Suppose $F(f)(x)\downarrow$. By induction on the generation of F can be proven that each computation tree z of $F(f)(x)$ is also a computation tree of $F(g)(x)$, for all applications of f and g used in the tree are the same. So $F(f)(x)\downarrow$ and $F(f)(x) = U(z) = F(g)(x)$.
compactness: Let $f \in \mathbf{P}$. If $F(f)(x)\downarrow$ then there is a computation tree z of $F(f)(x)$, which contains only finitely many applications of f. Regard the finite function u that is the same as f on input that is used in z and undefined everywhere else. For the same reason as above, z is also a computation tree of $F(f)(x)$. Thus $F(f)(x) = F(u)(x)$. □

Troelstra develops the floating product topology to derive similar results see [Troelstra73].
We did not introduce a notion of recursive relations. An attempt would be:
A relation $R \subseteq \mathbf{P} \times \mathbb{N}^n$ is recursive if its characteristic functional

$$\chi_R(f, \vec{x}) = \begin{cases} 1 & \text{if } (f, \vec{x}) \in R \\ 0 & \text{otherwise} \end{cases}$$

is recursive.
As χ_R is defined everywhere, there are no interesting recursive relations. In particular there is no recursive relation that can serve in the Normal Form Theorem. The application of theorem 2.1.14 proves this formally.

Theorem 2.1.15 *1. Let $F : \mathbf{P} \to \mathbf{P}$ be a monotone and compact. If*

$$\forall x \in \mathbb{N}, f \in \mathbf{P}[F(f)(x)\downarrow]$$

then F is independent of f, i.e there is a function $\psi : \mathbb{N} \to \mathbb{N}$ such that for all f, x

$$\psi(x) = F(f)(x).$$

If F is recursive then ψ is so.

2. *Every relation $R \subseteq \mathbf{P} \times \mathbb{N}$ whose characteristic functional is compact and monotone is trivial. This means there is a relation $S \subseteq \mathbb{N}$ such that for all f, x*

$$(f, x) \in R \iff x \in S.$$

If R is recursive then S is recursive as well.

PROOF:

1. If F is a partial recursive functional and $\forall x \forall f[F(f)(x)\downarrow]$ then $F(\emptyset)(x)\downarrow$. By monotony it follows that $F(f)(x) \simeq F(\emptyset)(x)$ for all f. Now, take $\psi(x) \simeq F(\emptyset)(x)$. If F is recursive, then it follows by proposition that 2.1.4 that $\psi(x)$ is partial recursive because $\emptyset$ is a partial recursive function.

2. The characteristic functional of a relation is defined everywhere. If R is recursive, $\chi_R = \psi$ for some recursive function ψ, which is the characteristic function of S.

□

In order to get more recursive predicates we can of course change the definition of recursive predicate. We should realize that with another definition the functional $F(f)(x) = \mu y[P(f)(x,y)]$ may turn out not to be recursive, even if P is recursive. We conclude that there is no notion of recursive predicate that makes sense.

2.1.3 Sequentiality

Another important property of partial recursive functionals is sequentiality, which we define and prove in a weaker formulation than usual [Barendregt84]. We write F_x for $\lambda f.F(f)(x)$.

Definition 2.1.16 A functional $F : \mathbf{P} \to \mathbf{P}$ is **sequential** if for all $x \in \mathbb{N}$

$$F_x \text{ is constant } \vee \exists n \in \mathbb{N} \forall f \in \mathbf{P}[f(n)\uparrow \implies F_x(f)\uparrow].$$

Sequentiality of a functional F means that there exists a sequential algorithm to evaluate $F(f)(x)$. If $F(f)(x)$ is not constant in f, the algorithm will use some values of f. Which values, may depend on f and on x, as in $f(f(x+1))$. If F is sequential one of the values of f will be inspected at first; in $f(f(x+1))$ this is $f(x+1)$. The first input of f, x+1 in the example, can not depend on any value of f. If f is undefined on this input, the algorithm that calculates $F(f)(x)$ will get stuck and $F(f)(x)$. will be undefined.
As opposed to parallel computations, it is not possible for sequential functionals to examine two values of f at the same time and take a decision based on the combined (termination) behaviour.

Example 2.1.17

1. The functional $F(f)(x) = f(f(x+1))$ is sequential: if $f(x+1)\uparrow$ then $F(f)(x)\uparrow$.

2. The functional

$$G(f)(x) \simeq \begin{cases} 0 & \text{if } f(4)\downarrow \vee f(5)\downarrow, \\ \uparrow & \text{otherwise} \end{cases}$$

is not sequential.

PROOF: Let $x \in \mathbb{N}$ It is clear that G_x is not constant. We have to show for all $n \in \mathbb{N}$ there is an $f \in \mathbf{P}$ such that $f(n)\uparrow$ and $G(f)(x)\downarrow$. Let $n \in \mathbb{N}$. If $n = 4$ then take $f = (5,0)$. If $n \neq 4$ then take $f = (4,0)$.

□

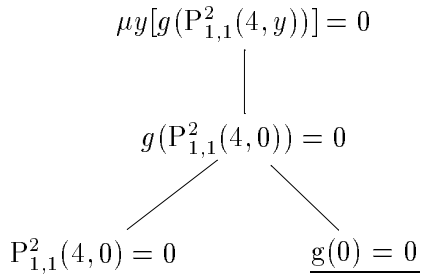
Theorem 2.1.18 *The partial recursive functionals are sequential.*

PROOF: We first give the intuition behind the proof.
Suppose $F : \mathbf{P} \to \mathbf{P}$ is a partial recursive functional, say $F = \Phi_e$. Let $f \in \mathbf{P}$ and $x \in \mathbb{N}$. Assume $\Phi_e(f)(x)\downarrow$. Examine the computation tree z of $\Phi_e(f)(x)$ by visiting the nodes in postfix order (that is, first visiting the subtrees of a node and then the node itself) until the first application of f appears. Say we come across $n_1, n_2, \ldots n_k$ and n_k represents the application of f to n. This n we are looking for in the definition of sequentiality.
By taking a compiler-like view on computation trees, we could say that input in a node is passed by its subtrees and by trees left from it. Therefore the input of node n_k is not influenced by any value of f.
We state: For all $g \in \mathbf{P}$ with $G(f)(x)\downarrow$

- the computation tree visited in postfix order starts with $n_1, n_2, \ldots n_{k-1}$.
- the k^{th} node represents the application $g(n)$.

So, $G(f)(x)\downarrow \implies g(n)\downarrow$, in other words $g(n)\uparrow \implies G(f)(x)\uparrow$.

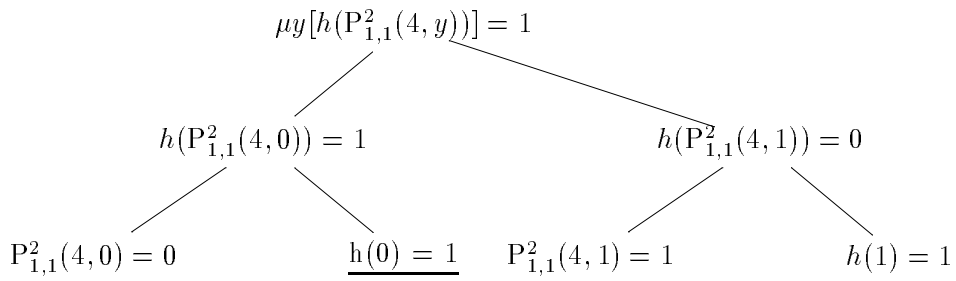
The example below illustrates this argument. Instead of encoded trees, we show plain trees. Take $F(f)(x) \simeq \mu\, y[f(\mathrm{P}^2_{1,1}(x,y)) = 0]$. Compare the computation trees of $F(g)(4)$ and $F(h)(4)$ where $g(x) = 0$ and $h(x) = \overline{sg}(x)$. The computation tree of $F(g)(4)$ looks like



If we flatten the tree and underline the first application of g, we get

$$<<\mathrm{P}^2_{1,1}(4,0) = 0>, \underline{<g(0) = 0>}, <g(\mathrm{P}^2_{1,1}(4,0)) = 0>,$$
$$<\mu\, y[g(\mathrm{P}^2_{1,1}(4,y))] = 0>>$$

The computation tree of $F(h)(4)$ is



Flattened and the first application of h underlined, we have

$$<<\mathrm{P}^2_{1,1}(4,0)=0>,\underline{<h(0)=1>},<\mathrm{P}^2_{1,1}(4,1)=1>,<h(\mathrm{P}^2_{1,1}(4,1))=0>,\\ <\mu\, y[h(\mathrm{P}^2_{1,1}(4,y))]=1>>$$

Indeed, in both flattened the first application of f occurs at the same place trees, the flattened trees are equal until the first application of f and the application is on the same input.
In order to turn this idea into a more precise argument, some auxiliary notions are introduced:
Recall that

- An encoded computation tree has the form: $<e, xs, \Phi_e(f)(x), t_1, \ldots, t_n>$, where $n \geq 0$, so leaves of the encoded tree have length 3.
- For the code of an application holds $(e)_1 = 3$.
- For all $z \in \mathbb{N}$ we have $(z)_0 = (z)_{1:0} = 0$.

Define

$$flat(z) \stackrel{\mathrm{d}}{=} \begin{cases} 0 & \text{if } z \notin I \\ z & \text{if } \lg z = 3 \;\&\; z \in I \\ flat(z)_4 \star \cdots \star flat(z)_{\lg(z)} & \text{otherwise} \\ \qquad \star <<(z)_1,(z)_2,(z)_3>> & \end{cases}$$

Then $flat(z)$ is the linear representation of z in postfix form. It is well-defined because $z < <z>$ for all $z \in \mathbb{N}$.
Define also

$$flattree(f)(e,x) \stackrel{\mathrm{d}}{=} flat(\mu[T(f)(e,<x>,z)]).$$

if $\Phi_e(f)(x)\!\downarrow$ then $flattree(f)(e,x)$ is its flattened computation tree.
Finally, define

$$firstapp(z) = \begin{cases} \mu i[(z)_{i,1,1} = 3] & \text{if } \exists i[\leq \lg z[(z)_{i,1,1} = 3], \\ 0 & \text{otherwise.} \end{cases}$$

Then, by induction on the generation of $e \in I$, we can prove: For all x, for all $e \in I$, for all f and $g \in \mathbf{P}$

$$\begin{aligned} \Phi_e(f)(x)\!\downarrow \implies & (flattree(f)(e,x))_{1:k} = (flattree(g)(e,x))_{1:k} \\ & \;\&\; (flattree(f)(e,x))_{k,1,1} = (flattree(g)(e,x))_{k,1,1} = 3 \\ & \;\&\; (flattree(f)(e,x))_{k,1,2} = (flattree(g)(e,x))_{k,1,2} \\ & \text{where } k = firstapp(flattree(f)(e,x)). \end{aligned}$$

Now we can prove sequentiality of partial recursive functionals easily. Let F be a partial recursive functional, suppose $F = \Phi_e$. Let x be in $\mathbb{N}$.

If F_x is not constant, then there is an $f \in \mathbf{P}$ such that $F(f)(x)\downarrow$. Let

$$z = flattree(f)(e,x),$$
$$k = firstapp(flattree(f)(e,x)),$$
$$n = (flattree(f)(e,x))_{k,1,2}.$$

We claim: $\forall g \in \mathbf{P}[g(n)\uparrow \implies F_x(g)\uparrow]$.
Suppose $g \in \mathbf{P}$ and $F(g)(x)\downarrow$. We prove $g(n)\downarrow$. Then

$$(flattree(g)(e,x))_{k,1,1} = 3,$$
$$(flattree(g)(e,x))_{k,1,2} = n,$$
$$(flattree(g)(e,x))_{k,1,3} = g(n).$$

We see $G(f)(x)\downarrow \implies g(n)\downarrow$. In other words: $g(n)\uparrow \implies G(f)(x)\uparrow$.
□

Application 2.1.19 The functional in example 2.1.17 is not partial recursive.

$$G(f)(x) \simeq \begin{cases} 0 & \text{if } f(4) \simeq 0 \vee f(5) \simeq 0 \\ \uparrow & \text{otherwise} \end{cases}$$

We remark that the restriction if G to $\mathbf{T}$, the set of total functions, *is* partial recursive: We can test $f(4)$ and $f(5)$ in any order without risk of nontermination.
The restriction to $\mathbf{PR}$, the set of partial recursive functions is not partial recursive. However, if the input function had been given as index e of φ_e, we would have been able to write down a partial recursive definition of g: Disposing of the code e, we can manipulate the computation of φ_e:

$$\psi(e,x) \simeq \begin{cases} 0 & \text{if } \exists z[T(e,<4>,z) \vee T(e,<5>,z)], \\ \uparrow & \text{otherwise.} \end{cases}$$

Then the function ψ is partial recursive and $\psi(e,x) \simeq G(f)(x)$. Such a function is called an effective operation. In chapter 4, we will investigate the relation between computable functionals and effective operations.

2.1.4 Do the partial recursive functionals capture the intuitive notion of computability on functions?

In the previous paragraph we found out that essentially parallel computations are not partial recursive. However, we may ask ourselves whether parallelism is conceptually computable, so whether the partial recursive functionals cover the intuitively computable functionals. Parallelism occurs in nature and in hardware: It is possible to put gates in parallel and then we can implement the functional in example 2.1.17 as an electronic circuit.
On the other hand, we may believe that we can overview only one process at the time. Although parallel evaluation of certain expressions is possible,

second order computability implemented in functional programming languages is in essence partial recursiveness. This notion seems easier to implement than effective computability, the second notion of computability.

Effective continuity is introduced in the following chapter. According to this notion, parallelism *is* computable. On total functions, where no problems with nontermination of input can arise, it coincides with partial recursiveness. So, since functions that represent real numbers are total, we do not have to choose.

2.2 Effective continuous functionals

A second attempt to formalize the notion of computable functional is based on an effective version of compactness and monotony. In 2.1 we argued that computable functionals should be compact: A terminating computation can use only finitely many values of the input function. It is also reasonable to require computable functionals to be monotone because an increase of information about the input function should yield as least as much information about the output. Of course, computable functions are computable on finite functions, that can be encoded as natural numbers.

On the other hand, if a functional meets the three requirements monotony, compactness and computability on finite functions, it can intuitively be computed: Provide the input function f by successively specifying new values about the input function and start computing on these finite approximations in parallel. If the functional is defined on f then, by compactness, this process will stop and it will produce the correct answer by monotony. As compactness and monotony can be described by a topology, namly the positive information topology, this notion of computability is called **effective continuity**. Because we can model parallelism, it is a proper extension of partial recursiveness. In order to focus on the essential ideas, rather than on complex formulation, we introduce these notions for functionals in $\mathbf{P} \to \mathbf{P}$. In chapter 2.3.2 we indicate how the theory can be generalized to functionals of arbitrary types. Before giving the definition of effective continuous functionals, two preparing sections about the positive information topology and finite functions respectively, are presented.

2.2.1 Compactness and monotony revised

The set $\mathbf{P}$ can be furnished with a topology, determined by the open basic open sets:

$$\widehat{u} = \{f \in \mathbf{P} \mid u \subseteq f\} \quad \text{where } u \text{ is a finite function.}$$

Proposition 2.2.1 *The set* $B = \{\widehat{u} \mid u \in \mathbf{P} \mid u \text{ finite}\}$ *indeed is the basis of a topology on* $\mathbf{P}$.

PROOF: $\mathbf{P} = \widehat{\emptyset} \in B$. Suppose $\widehat{u}, \widehat{v} \in B$. Then

$$\widehat{u} \cap \widehat{v} = \begin{cases} \emptyset & \text{if } \exists i \in \mathrm{Dom}(u) \cap \mathrm{Dom}(v)[u(i) \neq v(i)] \\ \widehat{u \cup v} = & \text{otherwise.} \\ \{f \in \mathbf{P} \mid u \subseteq f \ \& \ v \subseteq f\} & \end{cases}$$

So for all $p \in \widehat{u} \cap \widehat{v}$ there is a basic set U with $p \in U \subseteq \widehat{u} \cap \widehat{v}$. $\square$

Then open sets are unions of basic open sets. This topology is called the **positive information topology** because basic sets are characterized by a finite amount of positive information of the form $f(x) = y$. Now we can speak about continuous functionals in $\mathbf{P} \to \mathbf{P}$.

Remark 2.2.2 The positive information topology coincides with the Scott topology induced by the c.p.o. $(\mathbf{P}, \subseteq)$, see [Barendregt84] section 1.2 for an exposure of [Scott]. We use the result that in an algebraic c.p.o., the sets $\widehat{u}$ where u compact, form a basis of the Scott topology and the observation that the compact element are just the finite functions. Details cf. [Barendregt84]. Futhermore, the topology is homeomorphic to $\{\mathbb{N} \cup \{\uparrow\}\}^{\mathbb{N}}$. Here $\{\mathbb{N} \cup \{\uparrow\}\}$ has the flat topology, where the collection of open sets is $\mathcal{P}(\mathbb{N}) \cup \{\mathbb{N} \cup \{\uparrow\}\}$. The topology on $\{\mathbb{N} \cup \{\uparrow\}\}^{\mathbb{N}}$. is formed by taking the infinite product. The key to the homeomorphism is the fact that in the basic open sets of an infinite product topology only finite products appear and in the basis of the positive information topology only finite functions.

Theorem 2.2.3 *[[Odifreddi89], ch. II-4] A functional $F : \mathbf{P} \to \mathbf{P}$ is continuous if and only if it is monotone and compact.*

PROOF: A basic theorem from topology states that a function f is continuous if and only if the preimage $f^{-1}(U)$ is open for all *basic* open sets U. The characterization of open sets we will use here is: A is open if and only if for all $x \in A$ there is a basic open set U such that $x \in U \subseteq A$.
$\Longrightarrow$: Let F be continuous. Let $x \in \mathbb{N}$ and $f \in \mathbf{P}$. Assume $F(f)(x)\downarrow$ and $F(f)(x) \simeq y$. Consider the finite function $\{(x, y)\}$. Then $\widehat{u}$ is open and $F(f)(x) \in \widehat{u}$, which means $f \in F^{-1}(\widehat{u})$. By continuity $F^{-1}(\widehat{u})$ is open. Using the characterization above, there is a basic open set $\widehat{v}$ with $f \in \widehat{v} \subseteq F^{-1}(\vec{u})$ which implies:

1. $F(v)(x) \simeq y$ and $v \subseteq f$. Thus F is compact.
2. If $g \subseteq f$ then $g \in \widehat{v}$, so $F(g)(x) \simeq y$. Hence F is monotone.

$\Longleftarrow$: Assume F is monotone and compact. Let u be a finite function. We show $F^{-1}(\widehat{u})$ is open.
Suppose $f \in F^{-1}((\widehat{u}))$. This means $f \in F^{-1}(\widehat{u})$ or $u \subseteq F(f)$ In other words, for each pair $(x_i, y_i) \in u$ one has $F(f)(x_i) \simeq y_i$. Now from compactness follows for all $x_i \in \mathrm{Dom}(u)$ there is a function v_i with $F(v_i)(x_i) \simeq y_i$. Let $w = \bigcup_i v_i$.

Being a finite union of finite functions, w is a finite function itself. Because $v_i \subseteq f$ for all i we also have $w \subseteq f$, which means $f \in \widehat{w}$.
By monotony we have for all $(x_i, y_i) \in u$ and all $g \supseteq w \supseteq v_i$ that $F(g)(x_i) \simeq y_i$. Thus $u \subseteq F(g)$ for all $g \in \widehat{w}$. Hence $F(\widehat{w}) \in \widehat{u}$, i.e. $\widehat{w} \subseteq F^{-1}(\widehat{u})$. Summarizing, $f \in \widehat{w} \subseteq F^{-1}(\widehat{u})$. □

Corollary 2.2.4 *[Odifreddi89] All partial recursive functionals are continuous.*

2.2.2 Encoding finite functions

Finite functions are partial recursive, so they have a Gödel index in the sequence $\varphi_0, \varphi_1, \varphi_2, \ldots$ of all partial recursive functions.
However, the finite functions can be encoded as a natural number in such a way that we can compute more information from the indexes. Especially, the domain is computable from this index.
For instance the encoding function can be defined by (see 2.1.1)

$$\{(x_1, y_1)(x_2, y_2), \ldots (x_n, y_n)\} \mapsto <<x_1, y_n>, \ldots, <x_n, y_n>> \quad (n \geq 0).$$

The range of this function, that is the set of natural numbers that encode finite functions, is called $\mathbf{I_{fin}}$. Its elements are called **canonical indexes** for finite functions, in contrast to (Gödel) indexes. If $e \in I_{fin}$ then v_e denotes the finite function encoded by e.

Theorem 2.2.5 *([Rogers67], ch. 5) There exists a recursive function $t : \mathbb{N} \to \mathbb{N}$ that translates special indices to Gödel indices, i.e. for all $e \in I_{fin}$*

$$\varphi_{t(e)} = v_e,$$

but there is no partial recursive function $s : \mathbb{N} \to \mathbb{N}$ to do the converse, i.e. for all $e \in I$ and φ_e finite

$$v_{s(e)} = \varphi_e.$$

PROOF: Define

$$f(e, x) = ((e)_k)_2 \text{ where } k = \mu i \leq \lg(e)[(e)_{i,1} = x].$$

Recall that $()_i$ selects the i^{th} element from an encoded sequence and $(x)_i = 0$ if $i > \lg x$. Then f is partial recursive and we can obtain t by the S^n_m-theorem with $f(e, x) = \varphi_{t(e)}$.
For part two of the theorem, suppose we have a partial recursive function s such that $v_{s(e)} = \varphi_e$ for all $e \in I$ and φ_e finite. Define

$$F(e, x) = \begin{cases} \varphi_e(e) & \text{if } x = e \\ \uparrow & \text{otherwise.} \end{cases}$$

Then by the S^n_m-theorem there is a partial recursive function f such that $F(e, x) = \varphi_{f(e)}(x)$. Besides, $\varphi_{f(e)}$ is finite and $s(f(e)) = 0 \iff \varphi_{f(e)} = \emptyset \iff \varphi_e(e)\downarrow$. As s is total on elements of I, we have reduced K to the set $\{0\}$. This contradicts the undecidability of K. □

Proposition 2.2.6 *The domain of v_e is uniformly decidable in e, which means that the function $f : \mathbb{N}^2 \to \mathbb{N}$*

$$f(e,x) = \begin{cases} 1 & \text{if } x \in \mathrm{Dom}(v_e), e \in I_{fin}, \\ 0 & \text{otherwise,} \end{cases}$$

is partial recursive.

PROOF: Define

$$f(e,x) = \begin{cases} 1 & \exists i \leq \mathrm{lng}(e)[(e)_{i,1} = x], \\ 0 & \text{otherwise.} \end{cases}$$

□

2.2.3 Effective continuous functionals

Using the encoding of finite functions as a natural number, the behaviour of F on finite functions can be described by a function.

Definition 2.2.7 The **compactification** of a functional F is a function $h = h_F : \mathbb{N} \to \mathbb{N}$ defined by

$$h(e,x) \simeq \begin{cases} F(v_e)(x) & \text{if } e \in I_{fin}, \\ \uparrow & \text{otherwise.} \end{cases}$$

Note that it is possible to define the compactification as a total function. We did not do this for the sake of simplicity.
A monotone and compact or, equivalently, continuous functional F is determined by its behaviour on finite functions:

$$F(f) = \bigcup \{F(u) \mid u \text{ finite } \& \, u \subseteq f\},$$

so we can describe F in terms of its compactification.

Proposition 2.2.8 *If $F : \mathbf{P} \to \mathbf{P}$ is a compact and monotone functional and h is its compactification then*

$$F(f)(x) \simeq h(\mu e[v_e \subseteq f \,\&\, h(e,x)\downarrow], x).$$

PROOF: Obvious. □

This shows us that if a continuous functional has a computable compactification, it can intuitively be computed by means of parallelism.
Let $f \in \mathbf{P}$ and $x \in \mathbb{N}$. We write for the moment $f(i)\downarrow^k$ if $f(i)$ produces a result within k seconds. Let $u_{n,k} = \{(i, f(i)) \mid i < n \,\&\, f(i)\downarrow^k\}$. The $u_{n,k}$'s

are intuitively computable in f: Examine which of the $f(0), \ldots f(n)$ terminates within k seconds. In order to compute $F(f)(x)$, compute, in parallel,

$$F(u_{1,1})(x), F(u_{1,2})(x), \ldots$$
$$F(u_{2,1})(x), F(u_{2,2})(x), \ldots$$
$$\vdots$$

If some of the $F(u_{n,k})(x)\downarrow$, then by monotony $F(f)(x) = F(u_{n,k})(x)$. As F is compact, if $F(f)(x)\downarrow$ then $F(u_{n,k})(x)\downarrow$ for some $u_{n,k}$.

Definition 2.2.9 A functional $F : \mathbf{P} \to \mathbf{P}$ is **effectively continuous** if it is continuous and its compactification is partial recursive. Thus we dispose of a partial recursive function $\psi : \mathbb{N}^2 \to \mathbb{N}$ such that

$$\psi(e, u) \simeq F(\upsilon_e)(x).$$

From theorem 2.1.15 follows that the natural notion of effective continuous predicate does not make sense.

Theorem 2.2.10 *Every partial recursive functional is effectively continuous.*

PROOF: Let $F : \mathbf{P} \to \mathbf{P}$ be a partial recursive functional and let $F = \Phi_d$. Let $t : \mathbb{N} \to \mathbb{N}$ be a recursive function that translates special indexes to their Gödel numbers. Now define

$$\psi_d(e, x) \simeq U(\mu z[T_{1,1}(\phi_{t(e)})(d, <x>, z)].$$

Then $\psi_d(e, x) \simeq \Phi_d(\upsilon_e)(x))$ so and the it is the compactification of Φ_d. By the Substitution Property (2.1.6) ψ_d is partial recursive. □

Example 2.2.11 Revising example 2.1.13 we see that the functional $G : \mathbf{P} \to \mathbf{P}$ with

$$G(f)(x) = \begin{cases} 0 & \text{if } f(4)\downarrow \vee f(5)\downarrow, \\ \uparrow & \text{otherwise,} \end{cases}$$

is effectively continuous.

PROOF: From example 2.1.13 we know G is continuous. According to proposition 2.2.6 the domain of υ_e is decidable if we have $e \in I_{fin}$, so we may take

$$\psi(e, x) \simeq \begin{cases} 0 & \text{if } 4 \in \operatorname{Dom} \upsilon_e, \\ 0 & \text{if } 5 \in \operatorname{Dom} \upsilon_e, \\ \uparrow & \text{otherwise.} \end{cases}$$

Then ψ is partial recursive and it is the compactification of G. □

Here we see effective continuity is a proper extension of partial recursiveness.

Theorem 2.2.12 *There exists a effective continuous functional that is not partial recursive.*

2.3 Survey and generalization of results

2.3.1 Summary

The diagram below shows the main notions in this chapter and their relations.

notions of computability:	partial recursive	$\Longrightarrow$ $\Longleftarrow\!\!\!\!\!\!/\;\;$	effectively continuous
			$\Downarrow$
auxiliary notions:	compact & monotone	$\Longleftrightarrow$	continuous

2.3.2 Generalizing the results

Most of the results about computable functionals are formulated for functionals having type $\mathbf{P} \to \mathbf{P}$. Generalizing these to functionals of any type, only requires some natural extensions and somewhat more complex notations; no new ideas are needed. We give a short overview of the general definitions:

Definition 2.3.1 Let $m, k_1, \ldots, k_m, n \in \mathbb{N}$. Let $F : \mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_m}$.

The partial order $\subseteq$ on $\mathbf{P}$ can be extended to $\subseteq_k$ on $\mathbf{P}^k$ by:

$$f \subseteq_k g \stackrel{\mathrm{d}}{=} \mathrm{Dom}(f) \subseteq \mathrm{Dom}(g) \;\&\; \forall \vec{x} \in \mathrm{Dom}(f)\,[f(\vec{x}) = g(\vec{x})]\,.$$

Then $\subseteq_{\vec{k}}$ on $\mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_m}$ can be defined by pointwise $\subseteq_{k_i}$ on $\mathbf{P}_{k_i}$.

2. The basis of the **positive information topology** on $\mathbf{P}_k$ is just

$$\widehat{u} = \{f \in \mathbf{P}_k \mid u \subseteq f\} \quad \text{where } u \text{ is a finite function in } \mathbf{P}_k$$

and $\mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_m}$ can be equipped with the product topology.

3. The functional F is **monotone** if for all $\vec{f}, \vec{g} \in \mathrm{Fdom}(F)$

$$\vec{f} \subseteq \vec{g} \implies F(\vec{f}) \subseteq_{\vec{k}} F(\vec{g}),$$

and F is called **compact** if for all $x \in \mathbb{N}^k, f \in \mathrm{Fdom}(F)$

$$\text{if } F(\vec{f})(\vec{x}) \simeq z \text{ then } \exists u \in \mathbf{P}\,[u \text{ finite } \;\&\; F(\vec{u})(\vec{x}) \simeq z]\,.$$

4. We write $F_{\vec{x}} = \lambda \vec{f}.F(\vec{f})(\vec{x})$. Then F is said to be **sequential** if for all $\vec{x} \in \mathbb{N}^k$

$$F_{\vec{x}} \text{ is constant } \vee \exists i \exists \vec{n} \in \mathbb{N}^k \forall f \in \mathrm{Fdom}(F)[f_i(\vec{n})\!\uparrow \implies F_{\vec{x}}(\vec{f})\!\uparrow].$$

Encoding finite functions in $\mathbb{N}^k \to \mathbb{N}$, is done like encoding unary functions, namely by encoding $k+1$-tuples.

Definition 2.3.2 The **compactification** for a functional in $\mathbf{P}_{k_1} \times \cdots \times \mathbf{P}_{k_m}$ is F is a function $h : \mathbb{N}^{m+1} \to \mathbb{N}$ defined by

$$h(\vec{e}, \vec{x}) \simeq \begin{cases} F(v_{e_1}, \ldots, v_{e_m})(x) & \text{if } e_1, \ldots e_m \in I_{fin}, \\ \uparrow & \text{otherwise.} \end{cases}$$

A functional F is **effectively continuous** if it is continuous and its compactification is partial recursive.
Now we can prove generalizations of former theorems by similar, – but more complex in formulation – arguments.

Theorem 2.1.14 can be generalized to

Theorem 2.3.3 *All partial recursive functionals are monotone and compact.*

Theorem 2.1.18 becomes

Theorem 2.3.4 *All partial recursive functionals are sequential.*

Theorem 2.2.3 is extended to

Theorem 2.3.5 *A functional is compact and monotone if and only if it is continuous.*

As an extension of Theorem 2.2.10 we get

Theorem 2.3.6 *Every partial recursive functionals is effectively continuous.*

We will use these results later, especially the case $\mathbf{P}^k \to \mathbf{P}$, which is applicable to representations of real functions in several variables.

2.4 Restricted functionals

Functionals that model functions on real numbers need not to be applicable to all elements on $\mathbf{P}$, only on the subset of representations of real numbers. For instance such functionals have type $\mathbf{R}_{\text{int}} \to \mathbf{R}_{\text{int}}$. Representations of real numbers are total functions.
A functional that is applicable only to a subset $V \subseteq \mathbf{P}$ is called a **restricted functional**.[1] In this chapter we explore computability — both partial recursiveness and effective continuity — on restricted functionals. How can these be defined. Do the properties proven for non-restricted functionals also hold for restricted functionals? In particular we want to compare their power of computability.
We will see that the situation for restricted functionals is very much alike that for to non-restricted functionals. Some adaptations are needed in the treatment of effective continuity and compactness, since finite functions need not to be in V.

[1] In the literature the term restricted functional is sometimes used for the case $V = T$.

Concerning computability, it is clear that every computable functionals that is computable on $\mathbf{P}$ is also computable on a subset of it. Furthermore we will show the converse: Every computable restricted functional has a computable extension to $\mathbf{P}$. This means that we are not able to compute more functionals due to the fact we have more information about the input, for we know we are in V.
Finally, we will study the case $V \subseteq T$, V consists of total functionals, which occurs when describing real functions. Some more elegant formulations of existing properties are proven. The most important result is that for $V \subseteq T$ partial recursiveness coincides with effective continuity.
In this entire section V is a subset of $\mathbf{P}$. Just as in the previous section results can be generalized to functionals other types. This time the extension of the definitions is left to the reader.

2.4.1 Partial recursive restricted functionals

As an input function is viewed as an object whose values can be used in a computation, values of any function can be substituted for values of elements in V. Therefore partial recursive restricted functionals are nothing more than restrictions of partial recursive non-restricted functionals.

Definition 2.4.1

1. A functional $F : V \times \mathbf{P}_m$ is partial recursive if F is the restriction to V of some partial recursive functional.

2. A predicate $P \subseteq V$ is recursive if its characteristic functional is recursive.

Example 2.4.2 The representation of $+$ on interval representation is partial recursive:

$$Plus : \mathbf{R}_{\text{int}} \times \mathbf{R}_{\text{int}} \to \mathbf{R}_{\text{int}}$$
$$Plus(\alpha, \beta)(n) = <\alpha'(n) + \beta'(n), \alpha''(n) + \beta''(n)>.$$

In contrast to non-restricted predicates, there are non-trivial recursive restricted predicates. For instance, take $V = \mathbf{T}$ and let u be the finite function $\{(4,0),(5,0)\}$. Then the predicate

$$\{f \in \mathbf{T} \mid u \subseteq f\}$$

is partial recursive, for the functional

$$\chi : \mathbf{T} \to \{0,1\},$$
$$\chi(f)(=) \begin{cases} 1 & \text{if } f(4) = 0 \ \& \ f(5) = 0, \\ 0 & \text{otherwise} \end{cases}$$

is so.

2.4.2 Effective continuous restricted functionals

Definition 2.4.3 A functional $F : V \to \mathbf{P}$ is compactificable if for all $f \in \mathbf{P}, x \in \mathbb{N}$ there is a finite function u such that for all $g \in \widehat{u} \cap V$

$$F(f)(x) \simeq F(g)(x).$$

In words: all elements of $\widehat{u}$ to which F is applicable are mapped onto the same element.

Like compactness and monotony, compactificability can be described topologically. The topology on V is induced by the positive information topology $\mathbf{P}$. Open sets in this induced topology are intersections of open sets in $\mathbf{P}$ and V. A basis can be obtained by taking the intersection of basic open sets in $\mathbf{P}$ and V.

Theorem 2.4.4 *A functional $F : V \to \mathbf{P}$ is continuous if and only if it is compactificable.*

PROOF: $\Longrightarrow$: Assume F is continuous. Let $f \in V, x \in \mathbb{N}$. Assume $F(f)(x)\downarrow$ and $F(f)(x) \simeq y$. Consider the finite function $\{(x, y)\}$. Then $\widehat{u}$ is open in $\mathbf{P}$. and $F(f)(x) \in \widehat{u}$. By continuity $F^{-1}(\widehat{u})$ is open in V. Thus there exists a finite function v such that

$$f \in \widehat{v} \cap V \subseteq F^{-1}(\vec{u}),$$

which means $F(\widehat{v}) \subseteq \widehat{u}$. So for all $g \in \widehat{v} \cap V$ we have $F(g)(x) \simeq y$.
$\Longleftarrow$: Assume F is compactificable. We wish to prove for all finite u, $F^{-1}(\widehat{u})$ is open in V, in other words $F^{-1}(\widehat{u}) \cap V$ is open in $\mathbf{P}$.
Let $f \in V, u \in \mathbf{P}$ finite and suppose $f \in F^{-1}(\widehat{u}) \cap V$. Then $u \subseteq F(f)$ which means for each pair $(x_i, y_i) \in u$ we have $F(f)(x_i) \simeq y_i$. By F is compactificable it follows that, for all x_i, there is a finite function w_i with

$$\forall g \in \widehat{w_i} \cap V\, [F(g)(x_i) \simeq y_i] \qquad (*)$$

Define $w = \bigcup_i w_i$. Then $f \in \widehat{w}$, as $w_i \subseteq f$ for all i. From $(*)$ follows $F(\widehat{w} \cap V) \supseteq u$ so $\widehat{w} \cap V \subseteq F^u$. $\square$

Although finite functions need not to be in V, the entire behaviour of a compactificable restricted functional can be described using only finite functions. By encoding them we have a compactification again.

Definition 2.4.5 Let $F : V \to \mathbf{P}$ be a compactificable functional. A **compactification** of F is a partial function $h = h_F : \mathbb{N}^2 \to \mathbb{N}$ such that by for all $x \in \mathbb{N}$, for all e with $\widehat{v_e} \cap V \neq \emptyset$

$$h(e, x) \simeq y \iff \forall f \in \widehat{v_e} \cap V [F(f)(x) \simeq y]$$

Note that h needs not to be recursive since its behaviour is only prescibed on element in $\mathbb{N} \times V$.

Proposition 2.4.6 *Let $F : V \to \mathbf{P}$ be a compactificable functional and let h be a compactification of F. Then for all $f \in V$*

$$F(f)(x) \simeq h(\mu e[v_e \subseteq f \ \& \ h(e,x)\downarrow], x).$$

Definition 2.4.7 1. A functional $F : V \to \mathbf{P}$ is effectively continuous if it is continuous and it has a compactification that is partial recursive.

2. A predicate $P \subseteq V$ effectively continuous if its characteristic functional is effectively continuous.

An elementary lemma from topology states that the restriction of a continuous function is continuous but not its converse. We will prove that even the restriction of an effectively continuous function is effectively continuous and conversely.

Proposition 2.4.8 *If $F : \mathbf{P} \to \mathbf{P}$ be a compact and monotone functional and $h : \mathbb{N} \to \mathbb{N}$ is its compactification, then $F \restriction_V : V \to \mathbf{P}$ is compactificable and $h : \mathbb{N} \to \mathbb{N}$ is also a compactification of $F \restriction_V$.*

Here we see that the definition of restricted effective continuous functional coincides with the definition if we have $V = \mathbf{P}$. The following corollary shows that for restricted functionals also, partial recursiveness implies effective continuity. Remark that, contrary to non-restricted functionals, the compactification h_F of a restricted partial recursive functional F is not effective in the index of the functional.

Corollary 2.4.9 *Each partial recursive restricted functional is effectively continuous.*

PROOF: If $F : V \to \mathbf{P}$ is partial recursive by definition it is the restriction of a partial recursive functional $G : \mathbf{P} \to \mathbf{P}$, which is effectively continuous. Then also its restriction is effectively continuous. □

Theorem 2.4.10 *Any continuous restricted functional has a continuous extension. If F is effectively continuous it has a effectively continuous extension.*

PROOF: Let $F : V \to \mathbf{P}$ be a continuous functional and let h be a compactification of F, which meets

$$F(f)(x) \simeq h(\mu e[v_e \subseteq f \ \& \ h(e,x)\downarrow], x).$$

Now just apply this definition to *all* elements of $\mathbf{P}$ and define $G : \mathbf{P} \to \mathbf{P}$ by

$$G(f)(x) \simeq h(\mu e[v_e \subseteq f \ \& \ h(e,x)\downarrow], x).$$

It is not difficult to see that g is compact and monotone and therefore continuous.
If F is effectively continuous, there exists a partial recursive compactification for F, that is one for G. □

Theorem 2.4.11 ([Odifreddi89], ch. II-4) *There is a* total *effectively continuous functional that can not be extended to a* total *effectively continuous functional.*

PROOF: (suggested by [Odifreddi89]) Consider the restricted functional

$$F : \mathbf{PR} \to \mathbf{P},$$
$$F(f)(x) = f(\mu z[f(z) = \varphi_z(z)]),$$

which should be read as

$$= f(\mu z[T((z)_1, <(z)_1>, (z)_2) \ \& \ f(z)_1 = U(z)_2])_1,$$

Then F is obviously partial recursive and thus effectively continuous. And F is also total: Let $f \in \mathbf{PR}$ be total, say f has Gödel index e. Then $f(e) = \varphi_e(e)$. So $F(f)(c)\downarrow$.

Now assume F has a total effectively continuous extension $G : \mathbf{P} \to \mathbf{P}$. We derive a contradiction.

Let $x \in \mathbb{N}$ and consider the function

$$f(z) = \begin{cases} \varphi_z(z) + 1 & \text{if } z \in K, \\ 0 & \text{otherwise.} \end{cases}$$

Because G is total $G(f)(x)\downarrow$. Since G is effectively continuous, there is a finite function $u \subseteq f$ such that for all $g \in \widehat{u} \cap \mathbf{PR}$ we have $F(f)(x) \simeq F(g)(x)$. Let $m = \max(\mathrm{Dom}(u).)$

Define

$$g(z) = \begin{cases} g(z) & \text{if } z \leq m, \\ \varphi_a(a) & \text{otherwise,} \end{cases}$$

where $a = \mu a[a \in K \ \& \ \varphi_a(a) \neq G(f)(x)]$.

Then $u \subseteq g$ and g is a partial recursive, because g is constant except for finitely many values. Say $g = \varphi_d$. Summarizing $g \in \widehat{u} \cap \mathbf{PR}$, so $G(g)(x) = G(f)(x)$.

On the other hand, $G(g)(x) = F(g)(x) = g(\mu z[g(z) = \varphi_z(z)])$. Let $z_0 = g(\mu z[g(z) = \varphi_z(z)])$.

Suppose $z_0 \leq m$. then $g(z_0) = \varphi_{z_0}(z_0)$, but $g(z_0) = f(z_0) = \varphi_{z_0}(z_0) + 1$. contradiction.

So $z_0 > m$. Thus $g(z_0) = \varphi_a(a) \neq G(f)(x)$. However, we also have $G(f)(x) = G(g)(x)F(g)(x) = g(z_0)$. Contradiction.

We conclude there is no total effectively continuous extension of F to $\mathbf{P}$. □

2.4.3 Functionals restricted to total input

Now we look closer at the case $V \subseteq \mathbf{T}$, where V contains only total functions. In this case a different formulation for the property "compactificable" is possible, which the intuitionists know as the General Principle of Continuity.

Proposition 2.4.12 *Let $V \subseteq \mathbf{T}$. Let $F : V \to \mathbf{P}$ be a restricted functional. The following formulations are equivalent:*

1. *F is compactificable,*
2. $\forall f \forall x \exists m \forall g[f =_m g \implies F(f)(x) = F(g)(x)]$,
3. $\forall f \forall n \exists m \forall g[f =_m g \implies F(f) =_n F(g)]$.

PROOF:

$1 \implies 2$: Let $f \in V, x \in \mathbb{N}$. Because F has a compactification there is a finite function u such that for all $f \in \mathrm{Dom}(\widehat{u}) \cap V$

$$F(f)(x) \simeq F(g)(x).$$

Let $m = \max(\mathrm{Dom}(u))$. Then for all $g, f =_m g$ we have $g \in \widehat{u}$, so $F(f)(x) \simeq F(g)(x)$.

$2 \implies 1$: Supose 2. Let $f \in \mathbf{P}$ and $x \in \mathbb{N}$. Then we have an m such that $\forall g[f =_m g \implies F(f)(x) = F(g)(x)]$ Take $u = \{(0, f(0)), (1, f(1)), \ldots (m, f(m))\}$. Thus for all $g \in \widehat{u}$ we have F (f)(x) = F (g)(x)]. This is exactly 1.

$2 \iff 3$**:** easy.

□

Definition 2.4.13 Let F be a compactificable functional. Let $f \in V, x \in \mathbb{N}$. An m such that

$$\forall g[f =_m g \implies F(f)(x) = F(g)(x)]$$

is called a **modulus of continuity** of F at (f, x), an m such that

$$\forall g[f =_m g \implies F(f) =_n F(g)]$$

is called a **n-modulus of continuity** at f.

Theorem 2.4.14 *Let $V \subseteq \mathbf{T}$ and $F : V \to \mathbf{P}$. Then F is partial recursive if and only if it is effectively continuous.*

PROOF: $\Longleftarrow$: Let F be a restricted effectively continuous functional. Then it has a partial recursive compactification, say $h = \varphi_d$, such that (by Proposition 2.4.6) one has

$$F(f)(x) \simeq h(\mu e[v_e \subseteq f \ \& \ h(e, x)\downarrow], x).$$

As $\subseteq$ is decidable on $\mathbf{T}$, we can compute F by means of the T-predicate for functions:

$$F(f)(x) \simeq$$
$$\varphi_d(\mu k[\forall x \leq \max \mathrm{Dom}(v_e)[v_e(x) = f(x)] \,\&\, T(d, <(k)_1, x>, (k)_2)], x).$$

Recall that $(k)_1$ and $(k)_2$ select the constituents of the ordered pair k. $\square$

2.4.4 Summary

We give an overview of the properties of functionals we have defined until now and their relations.
In the diagram below the first line is about non-restricted functionals; the second about restricted. The "implication" $A \overset{\exists}{\Longrightarrow} B$ means If $F \restriction_V : V \to \mathbf{P}$ has property A then there is an extension of F to $\mathbf{P}$ with property B.

$\mathbf{P}$:	partial recursive	$\Longrightarrow$	effectively continuous	$\Longleftarrow$	continuous	$\Longleftrightarrow$	compact & monotone
	$\Downarrow \Uparrow \exists$		$\Downarrow \Uparrow \exists$		$\Downarrow \Uparrow \exists$		$\Downarrow \Uparrow \exists$
V :	partial recursive	$\Longrightarrow$	effectively continuous	$\Longleftarrow$	continuous	$\Longleftrightarrow$	compactificable

The "implications" $\overset{\exists}{\Longrightarrow}$ do not hold if exclusively total functionals are considered.

Chapter 3

Computability on representations of real numbers

Being equipped with a notion of computability on functions, we wish to explore which operations on representations of real numbers can be computed. This chapter is concerned with three kinds of operations on representation systems. Firstly, *computable approximations* are considered. We argued in 1.1.2 that a representation that is suited for implementation should have the effective approximation property: Given a representation α there must be an effective way to approximate $\pi(\alpha)$ upto any precision.
This means that there is a computable functional $A : V \times \mathbb{Q}^+ \to \mathbb{Q}$ that maps a representation and a positive distance q to a rational that is not more than q away from $\pi(\alpha)$.
Moreover we study *translations* between representation systems. Does a computable functional exist that translates an element in one system into a representation of the same real number in the other system? We prove that a system has the effective approximation property if and only if there is a computable translation to $\mathbf{R}_{\text{int}}$. Two system that can effectively be translated into each other will be called recursively equivalent.
Finally, we define a notion of *computable real functions* on a representation system. Unfortunately, the class of computable real functions depends on the representation. So there is no absolute notion of computable real function. We work out the computability of $+$, sin and $<$. Two recursively equivalent representations determine the class of computable real functions.
We treat the relations between the mentioned operations and explore the computability of these in the standard representation systems from 1.1.4, which will appear to be different from computational point of view. An important result is that in $\mathbf{R}_{\text{int}}$ all computable functions are continuous in the Euclidean topology.

This chapter is organized as follows. Section 3.1 defines the effective approximation properties and proves this property for the systems presented in 1.1.4

We work out the notions of computable translations and formulate all positive results in subsection 3.2.
Section 3.3 introduces computable real functions in $\mathbb{R}^n \to \mathbb{R}$ and negative results concerning translations are proved.
The reader is referred to 3.2.1 and to 3.3.1 respectively for a summary of the computable translations and functions.
Note that representations of real numbers are total functions, so the notions of partial recursiveness and effective continuity coincide by 2.4.14.

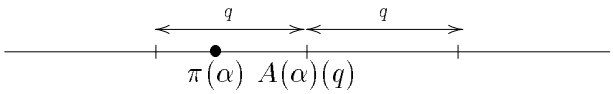
3.1 Approximations of real numbers

Definition 3.1.1 A representation system (V, π) has the **effective approximation property** if there exists a partial recursive functional, called an approximator of (V, π), $A : V \times \mathbb{Q}^+ \to \mathbb{Q}$ such that for all $\alpha \in V, q \in \mathbb{Q}^+$

$$|A(\alpha)(q) - \pi(\alpha)| < q.$$

In words: The distance between $A(\alpha)(q)$ and $\pi(\alpha)$ is less than q; Physicists also use the notation $\pi(\alpha) = A(\alpha)(q) \pm q$.

In a picture:



As announced in chapter 1.1.2 real numbers can effectively be approximated in the systems we consider.

Theorem 3.1.2 *The systems* $\mathbf{R}_{\text{int}}$, $\mathbf{R}_{\text{cauchy}}$, $\mathbf{R}_{B\text{-ary}}$, *and* $\mathbf{R}_{\text{ded}}$ *satisfy the finite approximation property.*

PROOF: We define an approximator for each system. The correctness is easily seen in most cases. Only $\mathbf{R}_{\text{ded}}$ is treated more elaborately.

- $\mathbf{R}_{\text{int}}$: Define $A_{\text{int}} : \mathbf{R}_{\text{ded}} \times \mathbb{Q}^+ \to \mathbb{Q}$

  $$A_{\text{int}}(\alpha)(q) = \alpha'(\mu n[\text{lng}(\alpha(n)) < q]).$$

  Note that $\text{lng}(\alpha(n)) = \alpha''(n) - \alpha'(n)$ is partial recursive in α and n.

- $\mathbf{R}_{\text{cauchy}}$: An approximator for Cauchy-sequences $A_{\text{cauchy}} : \mathbf{R}_{\text{cauchy}} \times \mathbb{Q}^+ \to \mathbb{Q}$ is defined by

  $$A_{\text{cauchy}}(\alpha, c)(q) = \alpha_{c(p-1)},$$
  where $p \in \mathbb{N}$ is such that $2^p < q$.

- $\mathbf{R}_{B\text{-ary}}$: Define $A_{B\text{-ary}} : \mathbf{R}_{B\text{-ary}} \times \mathbb{Q}^+ \to \mathbb{Q}$

$$A_{B\text{-ary}}(\alpha)(q) = \sum_{i=0}^{n_0} \frac{\alpha_i}{B^i},$$

$$\text{where } n_0 = \mu n \left[\frac{1}{10^n} < q \right].$$

- $\mathbf{R}_{\text{ded}}$: The key to the algorithm for Dedekind cuts is to "take samples of α" with steps of length q: If we find a $k \in \mathbb{Z}$ such that

$$k \cdot q \in \alpha \text{ and } (k+1) \cdot q \notin \alpha$$

we know $k \cdot q < \pi_{\text{ded}}(\alpha) \leq (k+1) \cdot q$, so

$$|k \cdot q - \pi_{\text{ded}}(\alpha)| \leq q.$$

Quite arbitrarily, we start looking at 0. If $0 \in \alpha$, we look for $k > 0$, if $0 \notin \alpha$, we search among negative k's.

Now define $A_{\text{ded}} : \mathbf{R}_{\text{ded}} \times \mathbb{Q}^+ \to \mathbb{Q}$ by

$$A_{\text{ded}}(\alpha)(q) = \begin{cases} q \cdot \mu k \left[(k+1) \cdot q \notin \alpha \right] & 0 \in \alpha, \\ -q \cdot \mu k \left[-k \cdot q \in \alpha \right] & 0 \notin \alpha. \end{cases}$$

□

It is more common to define a Cauchy sequence without mentioning the modulus of convergence. Then we get: A Cauchy sequence is a sequence $\alpha_0, \alpha_1, \ldots$ such that

$$\forall k \in \mathbb{N} \exists N \forall n, m \geq N \left[|\alpha_n - \alpha_m| < \frac{1}{2^k} \right].$$

We have not used this definition because it does not have the effectively approximation property. Although we know a Cauchy sequence has a limit, a prefix of a Cauchy sequence tells nothing about value of the limit.
Let S be the set of Cauchy sequences as meant above; then π_S takes the limit of a sequence.

Theorem 3.1.3 *The representation system (S, π_S) does not have the effective approximation property.*

PROOF: Assume $A : S \times \mathbb{Q}^+ \to \mathbb{Q}$ is a partial recursive approximator. Consider the zero function $\underline{0}$. Then $\pi_S(\underline{0}) = 0$, so $A(\underline{0})(\frac{1}{2}) \leq \frac{1}{2}$. Because A is partial recursive, it has a modulus of continuity at $(\underline{0}, \frac{1}{2})$, say m. Define

$$\alpha(n) = \begin{cases} 0 & \text{if } n \leq m, \\ 1 & \text{otherwise.} \end{cases}$$

Then $\alpha =_m \underline{0}$, so $A(\underline{0})(\frac{1}{2}) = A(\alpha)(\frac{1}{2}) \leq \frac{1}{2}$. Then

$$\left| A(\alpha)(\frac{1}{2}) - \pi_S(\alpha) \right| = \left| A(\alpha)(\frac{1}{2}) - 1 \right| > \frac{1}{2}.$$

This contradicts the assumption that A is an approximator. □

We may also wonder whether it is necessary to represent a Dedekind cut by its characteristic function, rather than by its "semi-characteristic function" χ_α

$$\chi_\alpha(q) = \begin{cases} 1 & \text{if } q \in \alpha, \\ \uparrow & \text{otherwise.} \end{cases}$$

The answer is, again, no because it is not possible to approximate a "semi-recursive" Dedekind cut effectively. The argument is the same as above: An approximator is not continuous. Remark that we have to do with *partial* functions.

3.2 Translations between representation systems

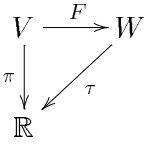
Definition 3.2.1 Let (V, π) and (W, τ) be representation systems of real numbers.

1. A **translation** from (V, π) to (W, τ) is a functional $F : V \to W$ such that for all $\alpha \in V$

   $$\tau(F(\alpha)) = \pi(\alpha).$$

   So a translation translates representations of a number in V to a representation of the same number in W.

   In a commuting diagram, we have

2. If there exists a partial recursive translation from V to W, we write: $(V, \pi) \preccurlyeq_r (W, \tau)$.

3. If both $(V, \pi) \preccurlyeq_r (W, \tau)$ and $(W, \tau) \preccurlyeq_r (V, \pi)$, we say that (V, π) and (W, τ) are **recursively equivalent**, which is denoted by $(V, \pi) \simeq_r (W, \tau)$.

Remark 3.2.2

- The relation $\simeq_r$ is clearly an equivalence relation The relation $\preccurlyeq_r$ is a preorder; $V \preccurlyeq_r W$ can be interpreted as "representations in V contain as least as much information as elements in W" because information in W can be computed from V.

- $$\left.\begin{array}{r} (V, \pi) \simeq_r (W, \tau) \\ \& \ (V, \pi) \preccurlyeq_r (S, \sigma) \end{array}\right\} \implies (W, \pi) \preccurlyeq_r (S, \sigma)$$

  and

  $$\left.\begin{array}{r} (V, \pi) \simeq_r (W, \tau) \\ \& \ (S, \sigma) \preccurlyeq_r (V, \pi) \end{array}\right\} \implies (S, \sigma) \preccurlyeq_r (W, \pi).$$

- A translation induces an isomorphism between the quotient structures $< V/\equiv_\pi, +_V, \cdot_V, \leq_V, 0_V, 1_V >$ and $< W/\equiv_\pi, +_W, \cdot_W, \leq_W, 0_W, 1_W >$. Here $+_V$, $\cdot_V$, $\leq_V$, 0_V, 1_V and $+_W$, $\cdot_W$, $\leq_W$ are representations of $+, \cdot, \leq, 0, 1$ in (W, τ) resp. (W, τ).

Theorem 3.2.3 *Let (V, π) be a representation system. Then (V, π) has the effectively approximation property if and only if $(V, \pi) \preccurlyeq_r (\mathbf{R}_{\text{int}}, \pi_{\text{int}})$.*

PROOF:
$\Longrightarrow$: Let A be a recursive approximator of V. The idea is to approximate a real number successively with $1, \frac{1}{2}, \frac{1}{4}, \ldots$ .Then we get a sequence of segments whose lengths converge to 0 and whose intersection is nonempty. The only thing to do is to indicate subintervals that are nested. Now define F and G by

$$F : V \to \mathbf{P},$$
$$F(\alpha)(n) = \langle A(\alpha)(\frac{1}{2^n}) + \frac{1}{2^n}, A(\alpha)(\frac{1}{2^n}) - \frac{1}{2^n} \rangle$$

and

$$G : V \to \mathbf{R}_{\text{int}},$$
$$G(\alpha)(0) = F(\alpha)(0),$$
$$G(\alpha)(n+1) = \langle \max(G(\alpha)(n)', F(\alpha)(n+1)'), \min(G(\alpha)(n)'', F(\alpha)(n+1)'') \rangle.$$

Then indeed $G(\alpha) \in \mathbf{R}_{\text{int}}$ and $\pi(\alpha) = \pi_{\text{int}}(G(\alpha))$.
$\Longleftarrow$: Let $F : V \to \mathbf{R}_{\text{int}}$ be translation from V to $\mathbf{R}_{\text{int}}$. Then $A : V \times \mathbb{Q}^+ \to \mathbb{Q}$ defined by $F(\alpha)(q) = \alpha'(\mu n[\text{lng}(\alpha(n)) < q])$ is an approximator. □

Corollary 3.2.4
- $\mathbf{R}_{\text{cauchy}} \preccurlyeq_r \mathbf{R}_{\text{int}}$,
- $\mathbf{R}_{B\text{-ary}} \preccurlyeq_r \mathbf{R}_{\text{int}}$ *for all bases B*,
- $\mathbf{R}_{\text{ded}} \preccurlyeq_r \mathbf{R}_{\text{int}}$,

Theorem 3.2.5 $\mathbf{R}_{\text{int}} \simeq_r \mathbf{R}_{\text{cauchy}}$.

PROOF: We must show $\mathbf{R}_{\text{int}} \preccurlyeq_r \mathbf{R}_{\text{cauchy}}$. The partial recursive functional

$$F : \mathbf{R}_{\text{int}} \to \mathbf{R}_{\text{cauchy}},$$
$$F(\alpha)(n) = \langle \alpha(n)', \mu k[\alpha(k) \leq \frac{1}{2^n}] \rangle,$$

translates an interval representation α into a Cauchy representation, for $\pi(\alpha) = \lim_{n\to\infty} \alpha'(n) = \pi(F(\alpha)(n))$. □

Theorem 3.2.6 *Let A and B be bases. If $\frac{1}{B}$ has a finite A-ary expansion, then $\mathbf{R}_{A\text{-ary}} \preccurlyeq_r \mathbf{R}_{B\text{-ary}}$.*

PROOF: We start with an example. Take $A = 2, B = 10$. Then $\frac{1}{2}$ has a finite decimal expansion for $[0.5]_{10} = \frac{1}{2}$. We write $e = [2.71828105904\ldots]_{10}$ in binary notation.

- step 1: Translate the part before the dot to binary representation, using the known algorithm: $[2]_{10} = [10]_2$.
- step 2: Translate the part behind the dot in the following way:

$$\begin{array}{rl}
\frac{1}{2} = 0.5 & /0.71828\ldots\backslash 1 \\
 & \underline{0.5} \\
\frac{1}{4} = 0.25 & /0.21828\ldots\backslash 0 \\
 & \underline{0} \\
\frac{1}{8} = 0.125 & /0.21828\ldots\backslash 1 \\
 & \underline{0.125} \\
\frac{1}{16} = 0.0625 & /0.10328\ldots\backslash 1 \\
 & \underline{0.0625} \\
 & /0.041\ldots \quad \backslash
\end{array}$$

Then each division needs only a finite part of the input $[0.71828\ldots]$. Indeed in the n^{th} one we need the same number of digits as $[\frac{1}{2^n}]_{10}$. Furthermore, we have

$$e = [10]_2 \cdot 1 + 1 \cdot \frac{1}{2} + 1 \cdot \frac{1}{4} + 0 \cdot \frac{1}{8} = [10.1101\ldots]_2$$

We stress that we work purely syntactical. If $\alpha = 0.999\ldots$ Then $[\alpha]_{10} = 1 = 2 \cdot 0.5$. However we proceed as follows:

$$\begin{array}{rl}
 & 0.5/0.999\ldots\backslash \mathbf{1} \\
 & \underline{0.5} \\
0.25 & /0.499\ldots\backslash \mathbf{1} \\
 & \underline{0.25}
\end{array}$$

Thus the algorithm will yield $0.111\ldots$ and indeed $[0.999\ldots]_{10} = [0.111\ldots]_2$. In general, if in the $\frac{1}{2^n}$ has k decimal digits n^{th} division, we do not consider more than k digits of the input.
Now, let A, B be arbitrary bases and let $\alpha \in \mathbf{R}_{A\text{-ary}}$. Find digits $d_0, d_1, \ldots$ by successive long division such that

$$[\alpha]_A - \sum_{i=0}^{n-1} \frac{d_i}{B^i} = d_n \cdot \frac{1}{B^n} + r_n$$

$$\text{with } r_n \leq \frac{1}{B^n} \text{ and } 0 \leq d_n < A - 1.$$

Then

$$[\alpha]_A = \sum_{i=0}^{\infty} \frac{d_i}{B^i} = [\alpha]_B.$$

The correctness of the long division algorithm ensures the existence of d_n, r_n with $r_n \leq \frac{1}{B^n}$. By induction follows $0 \leq d_n < A - 1$. □

We remark that, in fact, step 1 and step 2 can be combined into a single one. The traditional algorithm used in step 1 that transforms integers in B-ary representation into their $A-ary$ equivalents, starts finding the least significant digit. In step 2 we give the most significant one at first. This also works for integers: Search for the largest power of B that fits in the number to be transformed and subtract as many times as possible. For instance:

$$\begin{array}{l} 2^{11} = 2048/2718\backslash 1 \\ \quad\quad\ \underline{2048} \\ 2^{10} = 1024/0670\backslash 0 \\ \quad\quad\ \underline{0\ \ \ } \end{array}$$

However, the traditional algorithm is more efficient.

Theorem 3.2.7 *Let A, B be bases. If $\frac{1}{B}$ does not have a finite A-ary expansion, then $\mathbf{R}_{A\text{-ary}} \not\leftrightarrows_r \mathbf{R}_{B\text{-ary}}$.*

PROOF: Suppose α is a A-ary expansion of $\frac{1}{B}$ and $F : \mathbf{R}_{A\text{-ary}} \to \mathbf{R}_{B\text{-ary}}$ is a translation of $\mathbf{R}_{A\text{-ary}}$ to $\mathbf{R}_{B\text{-ary}}$.

Assume F is partial recursive. Then it is continuous. Examine $F(\alpha)(1)$, the first digit of α behind the dot. Then F has a 2-modulus of continuity at $(\alpha, 1)$, say m. Define $\alpha_0, \alpha_1 \in \mathbf{R}_{A\text{-ary}}$ by

$$\alpha_0(n) = \begin{cases} \alpha(n) & \text{if } n \leq m, \\ 0 & \text{otherwise} \end{cases}$$

and

$$\alpha_1(n) = \begin{cases} \alpha(n) & \text{if } n \leq m, \\ A-1 & \text{otherwise.} \end{cases}$$

Then $\alpha_0 =_m \alpha_1 =_m \alpha$, so

$$F(\alpha_1)(0) = F(\alpha)(0) = 0 \text{ and } F(\alpha_0)(1) = F(\alpha_1)(1).$$

On the other hand, because $\pi_{A\text{-ary}}(\alpha)$ is not a A-ary fraction,

$$\pi_{A\text{-ary}}(\alpha_0) < \pi_{A\text{-ary}}(\alpha) = \frac{1}{B} < \pi_{A\text{-ary}}(\alpha_1) \leq 1.$$

Thus

$$F(\alpha_0)(1) = 0.$$

Since $F(\alpha_1)(0) = 0$,

$$F(\alpha_1)(1) \geq 1.$$

Contradiction.

□

In order to formulate a simple condition equivalent to $\mathbf{R}_{A\text{-ary}} \preccurlyeq_r \mathbf{R}_{B\text{-ary}}$, we need an elementary number theoretic result.

Lemma 3.2.8 *Let A, B be bases.*

$$\frac{1}{A} \text{ is } B\text{-ary fraction} \iff \forall p, prime[p|A \implies p|B].$$

PROOF: $\implies$: Suppose $\frac{1}{A}$ is B-ary fraction, say $\frac{1}{A} = \frac{t}{B^n}$ with $t \in \mathbb{Z}, n \in \mathbb{N}$. Then $A \cdot B^n = t \in \mathbb{Z}$, so $A|B^n$ and therefore, for all primes p, if $p|A$ then $p|B$. $\impliedby$: For all primes p, if $p|B$ and $p \cdot t = B$ then $\frac{1}{p} = \frac{t}{B}$ is a B-ary fraction. If $p_1 \cdots p_n$ is a prime factorization of A, then $\frac{1}{A} = \frac{1}{p_1} \cdots \frac{1}{p_n}$. Thus $\frac{1}{A}$ is a product of B-ary fractions and therefore a B-ary fraction. □

Corollary 3.2.9 *For all bases A, B*

1. $\mathbf{R}_{B\text{-ary}} \preccurlyeq_r \mathbf{R}_{A\text{-ary}}$ *if and only if* $\forall p, prime\,[p|A \implies p|B]$.
2. $\mathbf{R}_{B\text{-ary}} \simeq_r \mathbf{R}_{A\text{-ary}}$ *if and only if* $\forall p, prime[p|A \iff p|B]$.

PROOF:

1. Use that q is a B-ary fraction if and only if it has a finite B-ary expansion.
2. from 1.

□

Theorem 3.2.10 *Let B be a base. Then $\mathbf{R}_{\text{ded}} \preccurlyeq_r \mathbf{R}_{B\text{-ary}}$.*

PROOF: Given $\alpha \in \mathbf{R}_{\text{ded}}$ we inductively construct the n^{th} digit of its B-ary equivalent.

- Firstly, we look for the part before the dot. Like in the approximator A_{ded} in the proof of theorem 3.1.2, we search a $k \in \mathbb{Z}$ such that

  $$k \in \alpha \text{ and } (k+1) \notin \alpha,$$

  Then $k \cdot B^0 = k < \pi_{\text{ded}}(\alpha) \leq k + 1 = (k+1) \cdot B^0$.

- Suppose we have k_n with

  $$k_n \cdot B^n < \pi(\alpha) \leq (k_n + 1) \cdot B^n.$$

  Now we look for k_{n+1} with $0 \leq k_{n+1} < B$ and

  $$\sum_{i=0}^{n} \frac{k_i}{B^i} + \frac{k_{n+1}}{B^{n+1}} \in \alpha \text{ and } \sum_{i=0}^{n} \frac{k_i}{B^i} + \frac{k_{n+1}+1}{B^{n+1}} \notin \alpha.$$

Such k_{n+1} exists, because

$$\frac{k_n}{B^n} \in \alpha \text{ and } \frac{k_n}{B^n} + \frac{(B-1)+1}{B^{n+1}} = \frac{k_n+1}{B} \notin \alpha.$$

By induction we can prove that for all $n \in \mathbb{N}$

$$\left| \pi_{\text{ded}}(\alpha) - \sum_{i=0}^{n} \frac{k_i}{B^i} \right| \leq \frac{1}{B^n},$$

So

$$\sum_{i=0}^{\infty} \frac{k_i}{B^i} = \pi_{\text{ded}}(\alpha),$$

thus $[k_0, k_1 k_2 \ldots]_B = \pi_{\text{ded}}(\alpha)$.

Now define

$$F(\alpha)(0) = \begin{cases} \mu k[k+1 \in \alpha] & \text{if } 0 \in \alpha, \\ -\mu k[-k \in \alpha] & \text{otherwise.} \end{cases}$$

$$F(\alpha)(n+1) = \mu k \left[\overline{F(\alpha)(n)}^n + \frac{k}{B^{n+1}} \in \alpha \ \& \ \overline{F(\alpha)(n)}^n + \frac{k+1}{B^{n+1}} \notin \alpha \right].$$

Recall that $\overline{\beta}^n = \sum_{i=0}^{n} \frac{\beta(i)}{B^i}$, so F in fact is defined by course-of-value recursion, cf. 2.1.7. □

We have seen that not all translations between the representation systems we consider are recursive. The next chapter provides some more examples. Surprisingly, the translations between the standard systems appear all to be recursive when restricted to representations of irrational number. However, the translations of representations of rationals is not always recursive. Before giving the proofs of these two statements, we introduce some notation.

Definition 3.2.11 The representations of rational and irrational numbers respectively are denoted by

$$\mathbf{Q}_{\text{int}} = \pi_{\text{int}}^{-1}(\mathbb{Q}),$$
$$(\mathbf{R} \setminus \mathbf{Q})_{\text{int}} = \pi_{\text{int}}^{-1}(\mathbb{R} \setminus \mathbb{Q}).$$

Similarly for $\mathbf{R}_{\text{ded}}$, $\mathbf{R}_{B\text{-ary}}$ and $\mathbf{R}_{\text{cauchy}}$.
The notions connected to recursive translation $\preccurlyeq_r$ and $\simeq_r$ naturally extend to subsystems $(V', \pi \restriction_{V'})$ of a representation system (V, π).

Lemma 3.2.12 $(\mathbf{R} \setminus \mathbf{Q})_{\text{int}} \preccurlyeq_r (\mathbf{R} \setminus \mathbf{Q})_{\text{ded}}$.

PROOF: The translation to Dedekind cuts is given by

$$F_{\text{ded}} : (\mathbf{R} \setminus \mathbf{Q})_{\text{int}} \to (\mathbf{R} \setminus \mathbf{Q})_{\text{ded}}$$
$$F_{\text{ded}}(\alpha)(q) = \begin{cases} 0 & \text{if } \exists n[q < \alpha'(n)], \\ 1 & \text{if } \exists n[q > \alpha''(n)]. \end{cases}$$

Since for $\pi(\alpha) \notin \mathbb{Q}$ we have, for all $q \in Q$

$$\exists n[q < \alpha'(n) \vee \alpha''(n) < q],$$

F_{ded} is partial recursive. Since for all $x \in \mathbb{R}$

$$\begin{aligned} x > \pi(\alpha) &\iff \exists n[x > \alpha''(n)] \text{ and} \\ x < \pi(\alpha) &\iff \exists n[x < \alpha'(n)] \end{aligned}$$

we have $\pi_{\text{ded}}(F(\alpha)) = \pi_{\text{int}}(\alpha)$. □

Corollary 3.2.13 $(\mathbf{R} \setminus \mathbf{Q})_{\text{cauchy}} \simeq_r (\mathbf{R} \setminus \mathbf{Q})_{\text{int}} \simeq_r (\mathbf{R} \setminus \mathbf{Q})_{B\text{-ary}} \simeq_r (\mathbf{R} \setminus \mathbf{Q})_{\text{ded}}$.

PROOF:

- $(\mathbf{R} \setminus \mathbf{Q})_{\text{cauchy}} \simeq_r (\mathbf{R} \setminus \mathbf{Q})_{\text{int}}$ follows by $\mathbf{R}_{\text{cauchy}} \simeq_r \mathbf{R}_{\text{int}}$.
- $(\mathbf{R} \setminus \mathbf{Q})_{\text{int}} \preccurlyeq_r (\mathbf{R} \setminus \mathbf{Q})_{\text{ded}}$ is proven 3.2.12 in above.
- $(\mathbf{R} \setminus \mathbf{Q})_{\text{ded}} \preccurlyeq_r (\mathbf{R} \setminus \mathbf{Q})_{B\text{-ary}}$ follows by $\mathbf{R}_{\text{ded}} \preccurlyeq_r \mathbf{R}_{B\text{-ary}}$, theorem 3.2.10.
- $(\mathbf{R} \setminus \mathbf{Q})_{B\text{-ary}} \preccurlyeq_r (\mathbf{R} \setminus \mathbf{Q})_{\text{int}}$ follows by $\mathbf{R}_{B\text{-ary}} \preccurlyeq_r \mathbf{R}_{\text{int}}$, theorem 3.1.2.

□

Theorem 3.2.14 *1.* $\mathbf{Q}_{\text{int}} \not\preccurlyeq_r \mathbf{Q}_{B\text{-ary}}$,

2. $\mathbf{Q}_{\text{int}} \not\preccurlyeq_r \mathbf{Q}_{\text{ded}}$.

PROOF:

1. (sketch) Let $F : \mathbf{Q}_{\text{int}} \preccurlyeq_r \mathbf{Q}_{B\text{-ary}}$be a translation. Assume that F is recursive. Consider the input function $\alpha(n) = \langle -\frac{1}{n}, \frac{1}{n} \rangle$. Let m be a modulus of F at α. Derive a contradiction using the functions

$$\alpha_0(n) = \begin{cases} \langle \frac{1}{B} - \frac{1}{n}, \frac{1}{B} + \frac{1}{n} \rangle & \text{if } n \leq m \\ \langle \frac{1}{B} - frac1m, \frac{1}{B} - \frac{1}{m} \rangle & \text{otherwise,} \end{cases}$$

and

$$\alpha_1(n) = \begin{cases} \langle \frac{1}{B} - \frac{1}{n}, \frac{1}{B} + \frac{1}{n} \rangle & \text{if } n \leq m \\ \langle \frac{1}{B} + \frac{1}{m}, \frac{1}{B} + \frac{1}{m} \rangle & \text{otherwise,} \end{cases}$$

2. Suppose $\mathbf{Q}_{\text{int}} \preccurlyeq_r \mathbf{Q}_{\text{ded}}$. By $\mathbf{R}_{\text{ded}} \to \mathbf{R}_{B\text{-ary}}$ it follows that $\mathbf{Q}_{\text{int}} \to \mathbf{Q}_{\text{ded}}$. Contradiction.

□

3.2.1 Summary

This diagram shows the recursive translations that were given in this section. The following chapter will show that other translations are not recursive.

$$\mathbf{R}_{\text{ded}} \xrightarrow[3.2.10]{} \mathbf{R}_{B\text{-ary}} \xrightarrow[3.2.4]{} \mathbf{R}_{\text{int}} \xleftrightarrow[3.2.5]{} \mathbf{R}_{\text{cauchy}}$$

$$\mathbf{R}_{B\text{-ary}} \xrightarrow[3.2.9]{\forall p, prime[p|A \Longrightarrow p|B]} \mathbf{R}_{A\text{-ary}}$$

3.3 Computable real functions via representation systems

Recall the definition 1.1.2: If (V,π) is a representation system for real numbers, and $X \subseteq \mathbb{R}$, then a (V,π)-representation of real function $f : X \to \mathbb{R}$ is a functional $F : V \to V$ such that $\pi(F(\alpha)) \simeq f(\pi(\alpha))$ for all $\alpha \in \pi^{-1}(X)$.
A (V,π)-representation of a partial real function $f : \mathbb{R}^n \to \mathbb{R}$ is a partial functional $F : V^n \to V$ such that $\pi(F(\alpha_1, \ldots, \alpha_n)) \simeq F(\pi(\alpha_1), \ldots \pi(\alpha_n))$. Possessing of a notion of computability on functionals we can speak of computable real functions on a representation.

Definition 3.3.1 Let (V,π) be a representation system of real numbers.

1. A real function $f : X \to \mathbb{R}$ is partial recursive with respect to (V,π) if there exists a partial recursive functional $F : V^n \to V$ that is a (V,π)-representation of f. The set of partial recursive real functions w. r. t. (V,π) is denoted by $\mathbb{R} \to_{(V,\pi)} \mathbb{R}$.

2. A predicate $p \subseteq \mathbb{R}^n$ is recursive if the characteristic functional of its (V,π)-implementation is recursive.

The definitions of recursive functions having type $f : \mathbb{R} \to \mathbb{Q}$, etc. are now straightforward.
The set of partial recursive sets depends on its representation. Recursively equivalent representation systems determine the same set of partial recursive functions.

Theorem 3.3.2 *For all representation systems* $(V,\pi), (W,\tau)$ *of real numbers*

$$(V,\pi) \simeq_r (W,\tau) \implies \mathbb{R} \to_{(V,\pi)} \mathbb{R} = \mathbb{R} \to_{(W,\tau)} \mathbb{R}.$$

PROOF: If $G : V \to W$ and $H : W \to V$ are recursive translations, and $F : V \to V$ is a partial recursive V representation of a real function f, then $H \circ F \circ G$ is a partial recursive W-representation of f. So $\mathbb{R} \to_{(V,\pi)} \mathbb{R} \subseteq \mathbb{R} \to_{(W,\tau)} \mathbb{R}$. Now $\supseteq$ follows by symmetry. □

Corollary 3.3.3 $\mathbb{R} \to_{(\mathbf{R}_{\text{int}})} \mathbb{R} = \mathbb{R} \to_{(\mathbf{R}_{\text{cauchy}})} \mathbb{R}$.

Theorem 3.3.4 *The function $+$ is partial recursive on $\mathbf{R}_{\text{int}}$ and on $\mathbf{R}_{\text{ded}}$, but not on $\mathbf{R}_{B\text{-ary}}$, for any base B.*

PROOF:

- Example 2.4.2 has shown that addition is partial recursive on interval representation.
- We stated in example 1.1.5 that the functional

$$Add_{ded}(\alpha,\beta)(q) = \begin{cases} 1 & \text{if } \exists p,r[\alpha(p)=1 \;\&\; \beta(r)=1 \;\&\; p+r=q], \\ 0 & \text{otherwise.} \end{cases}$$

  is a Dedekind representation of $+$. It equals

$$= \begin{cases} 1 & \text{if } \exists p,r[p \in \alpha \;\&\; r \in \beta \;\&\; p+r=q], \\ 0 & \text{if } \exists p,r[p \notin \alpha \;\&\; r \notin \beta \;\&\; p+r=q]. \end{cases}$$

  Since the clauses on the right hand side are mutually exclusive, and decidable Add_{ded} is recursive.

- The problem with $+$ on $\mathbf{R}_{B\text{-ary}}$ is that carries come from the right. From a finite prefix we are unable to predict whether one will eventually show up. Now, let B be a base.

  Suppose we have a recursive B-ary representation F of $+$. Consider $\alpha = 0.000\ldots$ and $\beta = 0.(B-1)(B-1)(B-1)\ldots$. Then $\pi_{B\text{-ary}}(\alpha) + \pi_{B\text{-ary}}(\beta) = 1$ and F has a modulus at $(\alpha,\beta,0)$, say m. Define

$$\alpha_{B-1} = \begin{cases} \alpha(n) & \text{if } n \leq m, \\ B-1 & \text{otherwise} \end{cases}$$

  and

$$\beta_0 = \begin{cases} \beta(n) & \text{if } n \leq m, \\ 0 & \text{otherwise.} \end{cases}$$

  then $\alpha_{B-1} =_m \alpha$ and $\beta_0 =_m \beta$. However $\pi_{B\text{-ary}}(\alpha_{B-1}+\beta) < 1 < \pi_{B\text{-ary}}(\alpha+\beta_0) < 2$, so $0 = F(\alpha,\beta_0)(0) = F(\alpha_{B-1},\beta)(0)$. Contradiction.

□

Corollary 3.3.5 *For all bases B, $\mathbf{R}_{\text{int}} \not\preccurlyeq_r \mathbf{R}_{B\text{-ary}}$ and $\mathbf{R}_{B\text{-ary}} \not\preccurlyeq_r \mathbf{R}_{\text{ded}}$.*

PROOF: From theorem 3.3.2 follows $\mathbf{R}_{\text{int}} \not\equiv_r \mathbf{R}_{B\text{-ary}}$. We have seen $\mathbf{R}_{B\text{-ary}} \preccurlyeq_r \mathbf{R}_{\text{int}}$ in 3.2.3. So we conclude: $\mathbf{R}_{\text{int}} \not\preccurlyeq_r \mathbf{R}_{B\text{-ary}}$. □

Theorem 3.3.6 *The function* sin *is partial recursive on $\mathbf{R}_{\text{int}}$ but neither on $\mathbf{R}_{B\text{-ary}}$, for any base B, nor on $\mathbf{R}_{\text{ded}}$.*

PROOF:

- $\mathbf{R}_{\text{int}}$: Our interval implementation of sin makes use of the Taylor series expansion of the sinus. For all $x \in \mathbb{R}$ and $n \in \mathbb{N}$

$$\left|\sin x - \sum_{k=0}^{\infty} t_n \cdot x^k k!\right| \leq \frac{1}{(n+1)!},$$

where $t_n = \sin^{(N)}(o)$, the n^{th} derivative of the sinus at 0. The sequence $t_0, t_1, \ldots$ equals 0,1,0,-1, ...

On the other hand, as its derivative is bounded, sin is a Lipschitz function: For all $x, y \in \mathbb{R}$

$$|\sin x - \sin y| \leq |x - y|.$$

These facts enable us to approximate $\sin x$, using only an approximation of x. Let $x \in \mathbb{R}$. Suppose we have an $a \in \mathbb{R}$ with

$$|x - a| \leq q$$

for some $q \in \mathbb{Q}$. Then for all $n \in \mathbb{N}$

$$\begin{aligned}
&\left|\sin x - \sum_{k=0}^{\infty} t_n \cdot a^k k!\right| \leq \\
&|\sin x - \sin a| + \left|\sin a - \sum_{k=0}^{\infty} t_n \cdot a^k k!\right| \leq \\
&|x - a| + \frac{1}{(n+1)} \leq \\
&q + \frac{1}{(n+1)}.
\end{aligned}$$

Now define

$$\begin{aligned}
&F : \mathbf{R}_{\text{cauchy}} \to \mathbf{R}_{\text{cauchy}}, \\
&F(\langle \alpha, c\rangle)(n) = \langle \sum_{k=0}^{\infty} t_n \cdot (\alpha(c(n)))^k k!, n+3\rangle.
\end{aligned}$$

Since

$$|\pi_{\text{cauchy}}(\alpha) - \alpha(c(n))| \leq \frac{1}{2^n},$$

we have

$$|\sin \pi_{\text{cauchy}}(\alpha) - F(\langle \alpha, c\rangle)(n))| \leq \frac{1}{2^n} + \frac{1}{(n+1)} \leq \frac{1}{2^{n+3}}$$

- $\mathbf{R}_{\text{ded}}$: The sinus does not have a recursive Dedekind representation: Here our notation is somewhat unfortunate. One should not confuse the real number π and the function π_{ded}. Suppose $F : \mathbf{R}_{\text{ded}} \to \mathbf{R}_{\text{ded}}$ of a representation of sin. Let α be a Dedekind representation of $1/6\pi$. For $\sin 1/6\pi = 1/2$, $F(\alpha)(1/2) = 0$.

  Assume F is partial recursive. We derive a contradiction. This time we work with finite functions instead of a modulus of continuity. As F is continuous, there is a finite function u such that

  $$\forall \beta \in \widehat{u} \cap \mathbf{R}_{\text{ded}}[F(\beta)(1/2) = F(\alpha)(1/2) = 0].$$

  There exists a $p \in \mathbb{Q}$ with
  - $1/6\pi < p < 1/2\pi$,
  - $p < q$, for all $q \in \text{Dom}(u)$ with $q \in \alpha$.

  Such a p exists because if $q \notin \alpha$ then $q > 1/6\pi$.
  Construct a function $\beta \in \mathbf{R}_{\text{ded}}$ with

  $$\beta(q) = \begin{cases} 1 & \text{if } q < p, \\ 0 & \text{otherwise.} \end{cases}$$

  Then $u \subseteq \beta$, so $F(\beta)(1/2) = 0$. However $\sin \pi_{\text{ded}}(\beta) > 1/2$, thus $F(\beta)(1/2) = 1$. Contradiction.

So sin is not partial recursive on $\mathbf{R}_{\text{ded}}$.

- $\mathbf{R}_{B\text{-ary}}$:

  Assume there is a recursive implementation $F : \mathbf{R}_{B\text{-ary}} \to \mathbf{R}_{B\text{-ary}}$ of the sinus. Let α be a representation of $\arcsin 1/B$. Because sin maps rational numbers unequal to 0, to irrational numbers[Siegel], $\arcsin 1/B$ is not a B-ary fraction. Say m is a modulus of F at $(\alpha, 1/B)$. Define

  $$\alpha_0(n) = \begin{cases} \alpha(n) & \text{if } n \leq m, \\ 0 & \text{otherwise.} \end{cases}$$

  and

  $$\alpha_1(n) = \begin{cases} \alpha(n) & \text{if } n \leq m, \\ B-1 & \text{otherwise.} \end{cases}$$

  Then $\alpha_0 =_m \alpha_1 =_m \alpha$, so $F(\alpha_0)(1) = F(\alpha_1)(1) = 0$. Furthermore $\pi_{B\text{-ary}}(\alpha)$ is irrational so

  $$0 \leq \pi_{B\text{-ary}}(\alpha_0) < \pi_{B\text{-ary}}(\alpha) < \pi_{B\text{-ary}}(\alpha_1) \leq 1.$$

  and therefore

  $$0 \leq \sin \pi_{B\text{-ary}}(\alpha_0) < \sin \pi_{B\text{-ary}}(\alpha) = \frac{1}{B} < \sin \pi_{B\text{-ary}}(\alpha_1) \leq 1.$$

  Then $F(\alpha_0)(=)0$ and $F(\alpha_1)(1) \geq 1$

Thus sin is not recursive.

□

Remark 3.3.7 Examining the first item of the proof, we observe that it can be mimicked to prove that other functions are computable on $\mathbf{R}_{\text{int}}$. Every function that has a computable Taylor series expansion , which means that the sequence of coefficients in the expansion is computable, and that has a bounded derivative, like the cosinus and the arctangent, is $\mathbf{R}_{\text{int}}$-computable.
A closer look may reveal more computable functions. Every function has a bounded derivative on a closed interval. If we can compute a bound from a given rational interval, we know up to which precision the input should be approximated. Then, if the function is computable on rational numbers, it is computable on all real numbers. Thus the tangent, log, e-power, etc. are computable. We do not go into details.

Corollary 3.3.8 $\mathbf{R}_{\text{int}} \not\leq_r \mathbf{R}_{\text{ded}}$.

Theorem 3.3.9 *1. The restriction of the entier function*

$$\lfloor\ \rfloor : \mathbb{R} \to \mathbb{Z}$$
$$\lfloor x \rfloor \leq x < \lfloor x \rfloor + 1.$$

to $\mathbb{R} \setminus \mathbb{Q}$ *is computable on* $\mathbf{R}_{\text{int}}$, $\mathbf{R}_{B\text{-ary}}$ *and* $\mathbf{R}_{\text{ded}}$.

2. Its restriction to $\mathbb{Q}$ *is not computable on any of these systems.*

PROOF:

1. We treat the system $\mathbf{R}_{B\text{-ary}}$, leavings the other cases to the reader. The functional $F : \mathbf{R}_{B\text{-ary}} \to \mathbb{N}$ defined by

   $$F(\alpha) = \lfloor \alpha(0) \rfloor$$

   is a B-ary representation of $\lfloor\ \rfloor$. Indeed, as $\alpha \notin \mathbb{Q}$ we have

   $$\alpha(0) < \pi(\alpha) < \alpha(0) + 1,$$

   so $\lfloor \pi(\alpha) = \alpha(0)$.

2. Suppose $\lfloor\ \rfloor$ is partial recursive on $\mathbb{Q}$. Examine a modulus m of an implementation of $\lfloor\ \rfloor$ at α, which is given by

   $$\alpha(n) = \begin{cases} 0 & \text{if } n = 0, \\ B-1 & \text{otherwise.} \end{cases}$$

   Then $\alpha_1(n) = \begin{cases} \alpha(n) & \text{if } n \geq m, \\ 0 & \text{otherwise,} \end{cases}$

   leads to problems.

□

Theorem 3.3.10 *Let $p \in \mathbb{Q}$. Then the real predicate $\{x \in \mathbb{R} \mid x < p\}$ is recursive on $\mathbf{R}_{\text{ded}}$.*

PROOF: The functional $F : \mathbf{R}_{\text{ded}} \to \{0, 1\}$

$$F(\alpha) = \begin{cases} 0 & \text{if } p \in \alpha, \\ 1 & \text{otherwise} \end{cases}$$

is a Dedekind representation of $\{x \in \mathbb{R} \mid x < p\}$. □

The theorem below show that the mentioned predicate is not recursive on $\mathbf{R}_{\text{int}}$.

Theorem 3.3.11 *The set $\mathbf{R}_{\text{int}}/\equiv_{\pi_{\text{int}}}$ with the quotient of the positive information topology according to $\equiv_{\pi_{\text{int}}}$ is homeomorphic to $\mathbb{R}$ with the Euclidean topology.*

PROOF: We show that 1. every open set in the quotient topology is also open in the Euclidean topology and 2. the converse. The homeomorphism then is the identity, for $(\mathbb{R}/\equiv_{\pi_{\text{int}}})$ *is* $\mathbb{R}$.

1. The set of open intervals with rational ends, is a basis of the Euclidean topology. We prove that these are all open in the quotient of the positive information topology. This means that their π_{int}- pre-image is open in the positive information topology. We write it as a union of open set in the positive information topology.

   Let $p, q \in \mathbb{Q}, p < q$. Then

   $$\begin{aligned} {\pi_{\text{int}}}^{-1}(p, q) &= \{\alpha \in \mathbf{R}_{\text{int}} \mid \alpha \text{ represents a number between } p \text{ and } q\} \\ &= \{\alpha \in \mathbf{R}_{\text{int}} \mid \exists n \in \mathbb{N}[p < \alpha'(n) \;\&\; \alpha''(n) < q]\} \\ &= \{\alpha \in \mathbf{R}_{\text{int}} \mid \exists n \in \mathbb{N} a, b \in \mathbb{Q}[p < a \;\&< q \;\&\; \alpha'(n) = a \\ &\qquad\qquad \&\; \alpha''(n) = b]\} \\ &= \cup_{n \in \mathbb{Q}, a, b \in \mathbb{Q}} \widehat{u_{nab}} \cap \mathbf{R}_{\text{int}} \end{aligned}$$

   where u_{nab} is the singleton function that maps n to the pair $\langle a, b\rangle$, so $u_{nab} = \{\langle a, b\rangle\}$.

2. Suppose that $U \subseteq \mathbb{R}$ is open in the quotient topology. This means that $\pi^{-1}(U)$ is open in the positive information topology. Write it as a union of basic p.i.-open sets: There are finite functions u_i such that:

   $$\pi^{-1}(U) = \cup_i \widehat{u_i} \cap V$$

   Let $x \in U$. We indicate a basic element I in the Euclidean topology, not necessarily one with rational ends, with $x \in I \subseteq U$.

   Choose a representation $\alpha \in \mathbf{R}_{\text{int}}$ of x which does not coincide with any of its ends. In other words, take $\alpha \in \mathbf{R}_{\text{int}}$ with $\pi_{\text{int}}(\alpha) = x$ and

$x \neq \alpha'(n)$ and $x \neq \alpha''(n)$ for all $n \in \mathbb{N}$. Then for all $n \in \mathbb{N}$ we have $x = \pi_{\text{int}}(\alpha) \in (\alpha'(n), \alpha''(n))$.

Because $\alpha \in {\pi_{\text{int}}}^{-1}(U) = \cup_i \widehat{u_i} \cap V$, we have a finite function u with $\alpha \in \widehat{u}$. Let $m = \max \text{Dom}(u)$. Then $\pi_{\text{int}}(\widehat{u}) = [u'(n), u''(n)] = [\alpha'(n), \alpha''(n)]$. Thus we have

$$x \in (\alpha'(n), \alpha''(n)) \subseteq [\alpha'(n), \alpha''(n)] = \pi_{\text{int}}(\widehat{u}) \subseteq U.$$

So take $I = (\alpha'(n), \alpha''(n))$.

□

Corollary 3.3.12 *Every $\mathbf{R}_{\text{int}}$-recursive real function is continuous in the Euclidean topology.*

Corollary 3.3.13 *1. Every $\mathbf{R}_{\text{int}}$-recursive function $F : \mathbb{R} \to \mathbb{N}$ is constant.*

2. Every $\mathbf{R}_{\text{int}}$-recursive predicate $P \subseteq \mathbb{R}^n$ is trivial.

It is open whether real functions that are $\mathbf{R}_{B\text{-ary}}$-recursive, are also Euclidean continuous.

3.3.1 Summary

This diagram indicates on which representation systems the functions $+$, sin and the relation < 0 are partial recursive. It also tells whether all partial recursive functions are continuous in the Euclidean topology.

	$+$	sin	< 0	continuous
$\mathbf{R}_{\text{int}}$=$\mathbf{R}_{\text{cauchy}}$	+	+	−	+
$\mathbf{R}_{\text{ded}}$	+	−	+	?
$\mathbf{R}_{B\text{-ary}}$	−	−	−	−

Chapter 4

computable real numbers and effective operations

Although it is not possible to represent all real numbers by a finite description, we can describe the elements of certain subsets by means of natural numbers. The fuctions in some representation system that are recursive can be represented by their Gödel indices. The real numbers we get this way are called **recursive real numbers** or computable real numbers.
Then we have a second manner – besides in terms of of functionals – to model functions on computable real numbers. That is as functions on natural numbers that respect the relation "represent the same computable real". If such a function is recursive we speak of an effective operation.
The sequel defines the notion of recursive real numbers relative to a representation system. It also shows that the sets of recursive real numbers coincide in the systems we consider. We will list some properties known from literature.
Concerning effective operations, it is easy to prove that every partial recursive functional on recursive real numbers corresponds to an effective operation. However we do not give a definitive answer to the converse problem.
Recall sequences, rational functions, etc. are all encoded as functions in $\mathbb{N} \to \mathbb{N}$, so it makes sense to speak of partial recursiveness.
Like before, **PR** is the set of partial recursive functions and **R** of the recursive functions. Furthermore the following notation is used. If $V \subseteq \mathbf{P}$ then V^c denotes the set of computable elements in V, thus $V^c = V \cap \mathbf{PR}$ and V^* denotes the set of Gödel numbers in V^c, i.e. $V^* = \{e \in \mathbb{N} \mid \varphi_e \in V^c\} = \{e \in \mathbb{N} \mid \varphi_e \in V\}$.

4.1 Computable real numbers

Definition 4.1.1 The set of **recursive elements** in a system (V, π) is just the V^c. We define the set of **recursive real numbers** with respect to (V, π), as the set $\pi(V^c)$, which are is real numbers that have a recursive representation in V.

Applying this definition is the system $\mathbf{R}_{\text{cauchy}}$ we get the following.

Example 4.1.2 The computabibity of a Cauchy sequence boils down to the computability of both the sequence and the modulus of convergence. We write $\mathbb{R}^c_{\text{cauchy}}$ for the set of computable Cauchy sequences.

Example 4.1.3 We show a computable representation of the number e in each system, by reconsidering the representations given in examples 1.1.5, 1.1.9, 1.1.11 and 1.1.13.

- $\mathbf{R}_{\text{ded}}$:

  We have seen that the function

  $$\eta(q) = \begin{cases} 1 & \text{if } q < e, \\ 0 & \text{otherwise} \end{cases}$$

  is a Dedekind representation of e. Using the Taylor series expansion of e, this function equals

  $$= \begin{cases} 1 & \text{if } \exists n[\sum_{k=0}^{n} \frac{1}{k!} - \frac{1}{(n+1)!} < q] \\ 0 & \text{if } \exists n[\sum_{k=0}^{n} \frac{1}{k!} + \frac{1}{(n+1)!} > q]. \end{cases}$$

  Since $e \notin \mathbb{Q}$, either the first or the last clause holds, so η is recursive.

- $\mathbf{R}_{B\text{-ary}}$:

  The digits of the B-ary representation of e can be computed by

  $$\eta(0) = 2$$
  $$\eta(n+1) = \lfloor B^n \sum_{k=0}^{B(n+1)} \frac{1}{k!} \rfloor - \lfloor B^n \sum_{k=0}^{B(n+1)} \frac{1}{k!} \rfloor B.$$

  We take the sum over the first $B \cdot n$ element, so ensure that we have enough information to determine the first n B-ary digits, which are given by $\lfloor B^n \sum_{k=0}^{Bn} \frac{1}{k!} \rfloor$. Then we retrieve the n^{th} one.

- $\mathbf{R}_{\text{int}}$:

  The interval representation of e we gave in example 1.1.11,

  $$\eta_{\text{int}}(n) = \langle \sum_{k=0}^{n} \frac{1}{k!} - \frac{1}{(n+1)!}, \sum_{k=0}^{n} \frac{1}{k!} + \frac{1}{(n+1)!} \rangle,$$

  is clearly recursive.

- $\mathbf{R}_{\text{cauchy}}$:

  And so is the Cauchy representation in example 1.1.13,

  $$\eta_{\text{cauchy}}(n) = \langle \sum_{k=0}^{n} \frac{1}{k!}, n+3 \rangle.$$

Theorem 4.1.4 *Let (V,π) and (W,τ) be representation systems of real numbers. Then*

1. *If $(V,\pi) \preccurlyeq_r (W,\tau)$ then $\pi(V^c) \subseteq \tau(W^c)$.*
2. *If $(V,\pi) \simeq_r (W,\tau)$ then $\pi(V^c) = \tau(W^c)$.*

PROOF:

1. The condition $(V,\pi) \preccurlyeq_r (W,\tau)$ expresses the existence of a recursive translation $F : V \to W$. By Proposition 2.1.4 it follows that F maps recursive functions to recursive functions. Therefore

   $$F(\mathbf{R} \cap V) \subseteq \mathbf{R} \cap W.$$

   Because F is a translation we have

   $$\pi(V \cap \mathbf{R}) = \pi(V^c) = \tau(F(V \cap \mathbf{R})) \subseteq \tau(W \cap \mathbf{R}) = \tau(W^c).$$

2. from 1.

□

Corollary 4.1.5 *1. $\mathbb{R}^c_{\text{int}} = \mathbb{R}^c_{\text{cauchy}}$.*

2. $\mathbb{R}^c_{\text{ded}} \subseteq \mathbb{R}^c_{B\text{-ary}} \subseteq \mathbb{R}^c_{\text{int}}$.

3. If a representation system of real numbers has the finite approximation property, then $\pi(V^c) \subseteq \mathbb{R}^c_{\text{int}}$.

PROOF:

1. and 2. are immediate

3. Theorem 3.2.3 states that the effective approximation property is equivalent to $V \preccurlyeq_r \mathbf{R}_{\text{int}}$.

□

The following – trivial – example shows that the effective approximation property does not ensure that $\mathbb{R}^c_{\text{int}} \subseteq V^c$.

Example 4.1.6 Examine the representation system

$$(\mathbf{R}_{\text{int}} \times \{H\}, \pi_{\text{int}} \circ \mathrm{P}^1_2).$$

Here H is a non-recursive function, for instance the characteristic function of the halting-problem and P^1_2 selects the second element from a pair. This system satisfies the effective approximation property, because the approximation can be computed using information from $\mathbf{R}_{\text{int}}$. However,

$$(\mathbf{R}_{\text{int}} \times \{H\})^c = \emptyset.$$

There are no computable elements in this system.

Of course, this is not very satisfactory. We would like to formulate an additional requirement for a system (V, π), that ensures $\mathbb{R}^c_{\text{int}} \subseteq \pi(V^c)$. Requiring $\pi(\mathbf{R}_{\text{int}}) \preccurlyeq_r V$ is too strong because, as we will see, $\mathbb{R}^c_{\text{ded}} \subseteq \mathbb{R}^c_{\text{int}}$ and we know $\mathbf{R}_{\text{int}} \not\preccurlyeq_r \mathbf{R}_{\text{ded}}$. Finding this requirement is still an open problem.

Proposition 4.1.7 *1.* $\mathbb{R}^c_{\text{int}} \subseteq \mathbb{R}^c_{\text{ded}}$,

2. $\mathbb{R}^c_{\text{int}} \subseteq \mathbb{R}^c_{B\text{-ary}}$ *for all bases* B.

PROOF: We prove the first statement in the proposition, the second is obtained by syntactic replacement("ded" by "B-ary"). We distinguish between rational and irrational real numbers in $\mathbf{R}_{\text{int}}$. Both are shown to be subsets of $\mathbf{R}_{\text{ded}}$.

- It is not difficult to show $\mathbb{Q} \subseteq \mathbb{R}^c_{\text{int}}$ and $\mathbb{Q} \subseteq \mathbb{R}^c_{\text{ded}}$, i.e. all rational numbers are computable in both systems.
- Theorem 3.2.14 states $((\mathbf{R} \setminus \mathbf{Q})_{\text{int}} \preccurlyeq_r (\mathbf{R} \setminus \mathbf{Q})_{\text{ded}}$. If F is a recursive translation, we have So, $\mathbb{R} \setminus \mathbb{Q} = \pi_{\text{int}}((\mathbf{R} \setminus \mathbf{Q})_{\text{int}}{}^c) \subseteq \pi_{\text{ded}}((\mathbf{R} \setminus \mathbf{Q})_{\text{ded}}{}^c)$.

We have $\mathbb{R}^c_{\text{int}} \subseteq \mathbb{R}^c_{\text{ded}}$. □

Corollary 4.1.8 *All common representation systems determine the same set of computable real numbers, i.e.*

$$\mathbb{R}^c_{\text{ded}} = \mathbb{R}^c_{\text{cauchy}} = \mathbb{R}^c_{B\text{-ary}} = \mathbb{R}^c_{\text{int}}.$$

The fact the standard representation systems yield the same set of computable reals, which we will denote by is $\mathcal{R}$, may indicate that with this set we have captured the intuitive notion of computable reals. This set has been studied in literature [Rice54][Mazur63][Bridges94]. It has nice properties, of which we list some without proof.

Proposition 4.1.9 (Rice) *The structure* $< \mathcal{R}, +, \cdot, \leq, 0, 1 >$ *is a totally ordered, real closed field. A real closed field is a field in which every polynomial with an odd degree and coefficients in* $\mathcal{R}$ *has a root in* $\mathcal{R}$ *and so has every polynomial* $x^2 - a$ *with* $a \in \mathcal{R}, a \geq 0$.

The recursive equivalent of completeness of $\mathbb{R}$, i.e. every Cauchy sequence of elements in $\mathbb{R}$ converges, has a recursive equivalent in $\mathcal{R}$. However the equivalent of the Bolzano-Weierstrass theorem does not hold. The notions of recursive sequence and (recursive) modulus of convergence defined for $\mathbb{Q}$ (cf. 1.1.12) can be extended to $\mathcal{R}$ easily.

Proposition 4.1.10 (Rice) *1. Every recursive, effectively converging sequence in* $\mathcal{R}$ *in has a limit in* $\mathcal{R}$.

2. Not every bounded recursive sequence in $\mathcal{R}$ *converges to a limit in* $\mathcal{R}$.

It is evident that $\mathcal{R}$ is countable. The proposition below shows that $\mathbb{R}^*_{\text{int}}$ is productive, which is a recursive equivalent of uncountability.

Proposition 4.1.11 $\mathbb{R}^*_{\text{int}}$ *is productive and* Π^0_2*-complete.*

PROOF: Productivity, is shown in [Bridges]. Π^2_0-completeness is easy, e.g. reduce $\mathbf{R}^*$, the index set of the (total) recursive functions to $\mathbb{R}^*_{\text{int}}$. □

The following theorem formulates a relation beween recusive reals and recursive real functions with repect to some represention system. In fact it provides a way to generate the recursive reals: by applying all recursive real functions to a, fixed, recursive real number. This does not contradict proposition 4.1.11 for the recursive real functions are not recursively enumerable either.

Theorem 4.1.12 *Let* (V, π) *be a representation system such that* $V^c \neq \emptyset$*. Let* $c \in \mathbb{R}^c_V$*. Then for all* $y \in \mathbb{R}$

$$y \in \mathbb{R}^c_V \iff \exists f \in \mathbb{R} \to_{(V,\pi)} \mathbb{R}[f(c) = y].$$

PROOF: $\Longrightarrow$: Let $y \in \mathbb{R}^c_V$. Then $\pi(\beta) = y$ for some $\beta \in \mathbf{PR}$. Define a real function by $f(x) = y$ for all x. Then $f(c) = y$ and f is partial recursive with respect to (V, π). Indeed that functional $F : V \to V$ with $F(\alpha) = \beta$ is an implementation of F and its is partial recursive by proposition 2.1.5. $\Longleftarrow$: Let $f \in \mathbb{R} \to_{(V,\pi)} \mathbb{R}$ with $f(c) = y$. Both f and y have a computable representation, which means that there exists a $\beta \in \mathbf{PR}$ with $\pi(\beta) = y$ and a recursive functional F such that $\pi(F(\alpha)) = \pi(\alpha)$ for all $\alpha \in V$. By proposition 2.1.4 the functional F maps $\mathbf{PR}$ to $\mathbf{PR}$, so $\pi(F(\beta)) = y \in \mathbb{R}^c_V$. □

4.2 Effective operations

We like to compare the two ways of representing functions on computable real numbers, by recursive functions and by recursive functionals that both preserve a notion of "represent the same real number."
Therefore, we wish to adapt an important result in the theory of recursive functions — sec, not representing real numbers — of Kreisel, Lacombe and Schoenfield. It states that the recursive ffuctions on Gödel numbers that preserve $\sim$ (represent the same recursive function) correspond to the recursive functionals on $\mathbf{R}$. Unfortunately, we did not succeed in this adaption. We have reformulated the theorem in our terminology.

Definition 4.2.1 The equivalence relation $\sim$ on $\mathbf{R}$ is defined by $e \sim e' \iff \varphi_e = \varphi'_e$.

Theorem 4.2.2 ([KLS59] in [Rogers67], ch 5.) *Let* $F : \mathbf{R} \to \mathbf{R}$ *be a restricted functional. Then* F *is effectively continuous if and only if there exists a recursive function* $\psi : \mathbf{R}^* \to \mathbf{R}^*$ *that preserves* $\sim$ *such that* $F(\varphi_e) = \varphi_{\psi(e)}$.

Now we wonder under what conditions this theorem can be generalized to other equivalence relations $\equiv_*$ than $\sim$ and other sets than $\mathbf{R}$. We are particularly interested in the case $(V, \equiv_\pi)$ where (V, π) is a representation system.

Definition 4.2.3 Let $V \subseteq \mathbf{R}$ and let $\equiv \subseteq V^2$ be an equivalence relation. An associate equivalence relation $\equiv_*$ on V^* is defined by:

$$e \equiv_* e' \iff \varphi_e \equiv_* \varphi'_e.$$

Note that is the relation $\equiv_*$ a refinement of $\sim$ and, like every equivalence relation, $\equiv$ is one of $=$.
A **effective operation** with respect to $\equiv_*$ is a recursive function $\psi : \mathbb{N} \to \mathbb{N}$ that respects $\equiv_*$, i.e.:

$$e \equiv_* e' \implies \psi(e) \equiv_* \psi(e').$$

Now the generalization of the theorem above can be formulated as:

1. Does every effective operation with respect to $\equiv_*$ induce an effectively continuous functional that respects $\equiv$?

2. Is every effectively continuous functional that preserves $\equiv$ the lifting of a effective operation with respect to $\equiv_*$?

The point of question 1. is the following: The effective operation ψ gets an index e as input. This is a finite object that describes φ_e as a whole. An effectively continuous functional may use only finitely many values of φ_e. It is therefore conceivable that a partial recursive functional has less computational power than an effective operation. However, the theorem of Kreisel, Lacombe and Schoenfield states that this is not the case if $V = \mathbf{R}$ and $\equiv_*$ is $\sim$; then ψ does not use more than finitely many values of φ_e. The proof of the converse — if F is partial recursive, there is recursive function $f : \mathbf{R}^* \to \mathbf{R}^*$ such that $F(\varphi_e) = \varphi_{f(e)}$ — is easy and immediately generalizes to other equivalence relations, providing an affirmative answer to question 2.

Theorem 4.2.4 *Let $V \subseteq P$ and let $\equiv$ be an equivalence relation on V. Let $F : V \to V$ be a effectively continuous functional that preserves $\equiv$. Then there exists an effective operation $\psi : \mathbb{N} \to \mathbb{N}$ with respect $\equiv_*$ such that, for all $e \in V^*$:*

$$F(\varphi_e) = \varphi_{\psi(e)}.$$

PROOF: [Odifreddi89] Let $F : P \to P$ be effectively continuous. Define

$$\varphi(e, x) = F(\psi_e, x).$$

Let h be the compactification of F. Then

$$F(\varphi_e, x) \simeq z \iff \exists d[v_d \subseteq \varphi_e \ \& \ h(d, x) \simeq z].$$

This the graph of φ is recursively enumerable, so φ is recursive. By the S^n_m-theorem, we can find a recursive function $f : \mathbb{N} \to \mathbb{N}$ such that

$$\varphi_{f(e)}(x) = \varphi(e, x) = F(\varphi_e, x).$$

□

The converse of this theorem, question 1, remains an open problem. We hope the proof of the theorem of Kreisel, Lacombe and Schoenfield can be adapted to our situation. Another open problem is the generalization to partial effective operations.

Bibliography

[Barendregt84] H.P. Barendregt *The Lamda Calculus: Its Syntax and Semantics.* North Holland, 1984.

[Beeson85] M. Beeson *Foundations of Constructive Mathematics.* Springer, 1985.

[BSS85] L. Blum, M. Shup, S.Smale *On a theory of computation and complexity over the real numbers and univeral machines.* Bull. Amer. Math. Soc.

[Bridges94] D.S. Bridges *Computability: a mathematical sketchbook.* Springer, 1994, pp 49 – 74.

[BR87] D.S. Bridges and F. Richman *Varieties of Constructive Mathematics.* Cambridge University Press, 1987.

[Grzegorczyk55] A. Grzegorczyk *Computable functionals.* In: Fund. Math., Vol 42, 1955, pp. 168–202.

[Grzegorczyk57] A. Grzegorczyk *On the definition of continuous computable real functions.* In: Fund. Math., Vol 44, 1957, pp. 61–71.

[Kleene59] S.C. Kleene *Recursive functionals and quantifiers of finite type.* In: Trans. Am. Soc., Vol 91, 1959, pp 1–52.

[KLS59] G. Kreisel, D. Lacombe, J.R. Schoenfield *Recursive functionals and effective operations.* In: A. Heyting, Constructivity in mathematics, North Holland 1959.

[Mazur63] Mazur *Computable Analysis* A. Grzegorczyk en H. Rasiowa, eds. Instytut Matematyczny Polskiej Akademii Nauk, Rozprawy Matematyczne 33 Warszawa : Panstwowe Wydawnictwo Naukowe, 1963

[Moschovakis65] Y.N.Moschovakis *Notation systems and recursive ordered fields.* In: Comp. Math., Vol 17, 1965, pp 40–71.

[Moschovakis69] Y.N.Moschovakis *Abstract higher order computability I.* In: Trans. Am. Soc., Vol 138 1969, pp 427–464.

[Odifreddi89] P. Odifreddi *Classical Recursion Theory.* North Holland 1989

[Peter51] R. Petèr *Recursive Funktionen* Akadémia Kiadó 1951, transl. American Press 1967.

[Platek66] R.A. Platek *Foundations of Recursion Theory.* Ph.D. Thesis, Stanford, 1966

[Pour-El] Marian B. Pour-El, Jonathan I. Richards *Computability in Analysis and Physics.* Springer-Verlag 1989

[Rice54] H.G. Rice *Recursive Real Numbers.* Proc. Am. Soc., Vol 5, 1954, pp 784–790.

[Robinson52] Robinson *Bespreking van het artikel van R. Petèr: 'Rekursive Funktionen Akad Kiadp, Budapest 1952'* In: The Journal of Symbolic Logic Vol 16, 1951 pp 280 ev

[Rogers67] Hartley Rogers, Jr *Classical Recursion Theory of Recursive Functions and Effective Computability.* The MIT press 1967

[Scott] D.S. Scott *Continuous lattices.* In F,.W. Lawvere, Toposes, Algebraic geometry and Logic, 274 Lecture Notes in Mathematics, Springer-Verlag.

[Siegel] Carl Ludwig Siegel *Trancedental numbers.* Princeton university press 1949, §11, final remark.

[Troelstra73] A.S. Troelstra (ed.) *Metamathematical investigations of intuitionistic arithmetic and analysis.* second, corrected edition, 1993, first edition 1973 ILLC Prepublication series X-93-05, University of Amsterdam pp. 147 e.v.

[Turing36] A.M. Turing *On computable real numbers with an application to the Entscheidungsproblem.* London Math. Soc., Vol 42, 1936, pp 230 –265, also in: Davis, M, The Undecidable, Raven Press, 1965.