UNIVERSITÀ DEGLI STUDI DI PISA
DIPARTIMENTO DI INFORMATICA

DOTTORATO DI RICERCA IN INFORMATICA

Università di Pisa-Genova-Udine

Ph.D. Thesis: TD - 6/93

A Functional Approach to Computability on Real Numbers

Pietro Di Gianantonio

Abstract.

The aim of this thesis is to contribute to close the gap existing between the theory of computable analysis and actual computation. In order to study computability over real numbers we use several tools peculiar to the theory of programming languages. In particular we introduce a special kind of typed lambda calculus as an appropriate formalism for describing computations on real numbers. Furthermore we use domain theory, to give semantics to this typed lambda calculus and as a conseguence to give a notion of computability on real numbers. We discuss the adequacy of Scott-Domains as domains for representing real numbers. We relate the Scott topology on such domains to the euclidean topology on $\mathbb{R}$. Domain theory turns out to be useful also in the study of higher order functions. In particular one of the most important results contained in this thesis concerns the characterisation of the topological properties of the computable higher order functions on reals. Our approach allows moreover to phrase and discuss w.r.t. real numbers issues of programming languages. We address the problem of defining an implementation of real numbers as an abstract data type. Finally we investigate algorithms for carrying out efficient computations on reals.

March 1993

ADDR:dip. di Matematica e Informatica, via delle Scienze, 33100 Udine Italy.
TEL:+39-432-558476 E_MAIL: pietro@dimi.uniud.it

Contents

Chapter 1

Introduction

Turing in 1937 was the first to introduce the notion of computable real number [32]. Since then a great number of different approaches have been used to investigate from a constructive standpoint main concepts arising is Analysis, such as real number, limit, derivative and measure. These enterprises have been referred to with various names e.g. recursive analysis, constructive analysis and computable analysis.

An important aspect which differentiates these approaches is the kind of logic being used: e.g. classical, intuitionistic, constructive.

Several authors give a completely constructive presentation of Analysis, using for example intuitionistically based logic. In these contexts all the definitions have a constructive meaning and only constructive methods are allowed in the proofs, Aberth [1], Beeson [4], Bishop [5] [9], Troelstra and van Dalen [31].

Particularly important is the work of Bishop [5] where a surprisingly large part of Analysis is shown to be feasable constructively. This result is obtained through a careful constructive "redefinition" of all the notions used, starting from the very fundamental ones e.g. that of real number and real function. Bishop's motto is: " mathematical statements should have numerical meaning". The constructive logic used by Bishop is not precisely defined. For instance Bishop uses freely the abstract concept of "rule" and of "operation" without giving them a mathematically precise meaning. This could have been done, for instance, by reformulating these concepts using Recursive Function theory. This is done by Aberth [1] who bases his work on a precise notion of algorithm. Other authors reformulate Analysis in the context of intuitionistic logic [31].

An alternative, and less radical, approach is that of using classical logic. In this case one takes as already given in a classical form the definition of real number and all the other concepts of Analysis Then using Recursion Theory one defines a countable subset of the real numbers, called *computable* (or recursive) *real numbers*. Informally a computable real number is a real number that can be defined by a recursive function.

Similarly one gives then a notion of *computable real function* and of *decidable relation on real numbers*.

The main results in this line of research are: the formulation of different definitions of computable real number, the comparison of different notions of computability, the relation between computability and continuity of functions. There are also many undecidability and complexity results.

Also in this line of research there is a plethora of different approaches. A key point of difference among these approaches is the notion of computability used. Turing machines are used in Ko and Friedman n [12], [13], Turing [32], Wiedmer [36]. A theory of computability of Baire spaces called "type 2 recursion theory" is used to study computability on real number in Kreitz and Weihrauch [14]. Finally approximation spaces are introduced and used in Lacombe [15], Martin-Löf [16], D. Scott [26], [26], Weihrauch [34].

1.1 Motivations for this work and results obtained.

Although the theory of computable Analysis can be considered a well developed subject, there have been so far very few attempts of implementing computable Analysis on digital computers [6], [7], [10], [33]. Such implementations should lead to the realization of "exact real number computation".

In ordinary practice the computation on real numbers is performed by approximating real numbers by a subset of the rational numbers and by approximating the arithmetic on real numbers by a limited precision arithmetic on rationals.

In exact real computation instead the result of a computation can be obtained with arbitrary precision and thus getting rid of the unfortunate phenomenon of the "round of error".

In this thesis we do not face directly the problem of defining a feasible effective implementation of exact real number computation. We go instead towards the direction of closing the gap existing between the theory of computable Analysis and actual computation.

In the thesis in order to study computability over real numbers we use several tools peculiar to the theory of programming languages.

In particular we introduce a special kind of typed lambda calculus as an appropriate formalism for describing computations on real numbers. Furthermore we use domain theory, to give semantics to this typed lambda calculus and as a conseguence to give a notion of computability on real numbers.

This approach turns out to be very fruitful for several reasons.

We discuss the adequacy of Scott-Domains as domains for representing real numbers. In the literature on real numbers computation different kinds of partial orders have been employed.

We relate the Scott topology on such domains to the euclidean topology on $\mathbb{R}$.

Using the theory of effective Scott domains we obtain simpler proofs of some of the classical results of constructive Analysis.

Domain theory turns out to be useful also in the study of higher order functions. In particular one of the most important results contained in this thesis concerns the characterisation of the topological properties of the computable higher order functions on reals.

Our approach allows moreover to phrase and discuss w.r.t. real numbers issues of programming languages. We address the problem of defining an implementation of real numbers as an abstract data type. That is we try to introduce a finite set of primitive functions on real number that can generate all the other computable functions.

Another problem that we address is the adequacy of using only sequential computation to express all the computable functions on reals. We give a partial answer to the question of whether parallel computations are more expressive.

Finally we investigate algorithms for carrying out efficient computations on reals. In particular we develop algorithms for the arithmetic operations which proceed contrary to the ones people normally use and children learn at school, i.e. from the more significant digits to the less significant ones. This is interesting in itself.

1.2 Outline of this thesis

In chapter 2 we give a survey of the different forms of real number representations used in computable Analysis.

In chapter 3 we present the syntax and the reduction rules of a simple typed lambda calculus. The various forms of real number representation are translated into this language. That is we show how to denote real numbers with terms of this language. In this way we make possible to define a notion of computability for real number and real functions.

In chapter 4 we present a denotational semantics for the language of chapter 3. Scott-Domains are used for this purpose. We introduce a second definition of computability on real numbers and we compare the two definitions. We deal then with the denotational and operational semantics of this language. Typical issues of the theory of the programing languages arise here. The language defined in the thesis it is not "complete". The language, in fact, is not sufficiently rich to denote all computable functions contained in the semantics domain. The question arises as to whether the language is sufficiently rich to denote all computable functions on reals that can be defined via domain theory. A partial answer is given.

In chapter 5 we present a new domain that can be used to study real numbers. This domain follows the approach of constructing approximation spaces for the real numbers. This construction has a domain-theoretic interest. In fact it is the first example of the use of Scott-domains in a problem area where normally continuous cpo's (i.e. continuous images of algebraic cpo's) are used.

In chapter 6 we investigate the connection between the Scott-topology and the topology on the real line. Moreover we present several important and original results which describe the topological properties of the computable real functions. The interest of these results lies in the possibility of characterizing the topological properties of the higher order functions.

In chapter 7 we discuss the problem of defining an abstract data type for the real numbers. In particular we address the problem of defining a finite set of primitive functions on real numbers such that any other computable function on reals can be obtained from them. A possible solution to this problem is given.

In chapter 8 we present two new representations for real numbers that can be fruitfully employed to improve the efficiency of some implementations of the real number computation.

The first one is an elegant variant of the "binary digit" representation. This representation leads to surprisingly simple algorithms for arithmetic operations.

The second one is a "digit representation" making use of large digits. This representation can be useful if we want to exploit fully the efficiency of the implementation of integer arithmetic which is hard wired in all modern computers.

Chapter 2

Real Number Representations

Since the seminal work of Turing, a great number of different approaches have been used to study constructive analysis. An important difference between these approaches lies in the way real numbers are represented. Different representations already occur in classical analysis: Cauchy sequences of rational numbers, Cauchy sequences of dyadic rationals, Dedekind cuts in the field of rationals, infinite decimal expansions, and so on. Classically all these representations are equivalent and we can study Analysis without worrying about which representation for real numbers we are currently using. Also in computable Analysis many of these representations turn out to be equivalent. But there are also some exceptions: for instance Dedekind cuts and Cauchy sequences turn out not to be equivalent.

Between the various constructive representations of real numbers in use there is one that can be consider the most general and taken as reference. In this representation a real number is defined as the limit of a computable sequence of rational intervals.

Definition 1 *A* rational-interval representation *of a real number $r \in \mathbb{R}$ is given by a computable sequence of intervals,*
$s_0 = [a_0, b_0], \ldots, s_i = [a_i, b_i], \ldots$
such that:
i) the end-points a_i, b_i of each interval s_i are rational numbers, that is s_i is a rational interval,
ii) $s_{i+1} \subseteq s_i$,
iii) $\lim_{i\to\infty}(b_i - a_i) = 0$
iv) $r = \bigcap_{i\in\mathbb{N}} s(i)$.

This representation has been used by several authors which dealt with the real number computability [15], [16], [26], [34]. It can be considered for many aspects, the general form of real representation. Many other representations

proposed in the literature differ from this one only in that they make use of a subset of the convergent sequences of rational intervals. Here are some examples:

Definition 2 *a) a real number r is represented by a computable Cauchy sequence of rational numbers $s_0, \ldots, s_i, \ldots$ and by a function $q : I\!N \to I\!N$ defining the convergence rate of the Cauchy sequence such that:*
i) $\forall i.j.k \mid s_{q(i)+j} - s_{q(i)+k} \mid \leq 2^{-i}$
ii) $r = \lim_{i\to\infty} s(i)$

b) a real number r is represented by a computable Cauchy sequence of rational numbers $s_0, \ldots, s_i, \ldots$ having a fixed rate of convergence:
i) $\forall i.j. \mid s_i - s_{i+j} \mid \leq 1/i$
ii) $r = \lim_{i\to\infty} s(i)$

c) corresponding to every natural number $p > 1$, we have the following form of real representation: a real number r is represented by a computable sequence of integer numbers $s_0, \ldots, s_i, \ldots$ such that:
i) $\forall i. \mid p \times s_i - s_{i+1} \mid < p$
ii) $r = \lim_{i\to\infty} s(i)/p^i$

d) given a natural number $b > 1$ a negative-digit representation with base b of a real number r is given by a sequence of integers $s_0, \ldots, s_i, \ldots$, such that:
i) $\forall i \in I\!N^+. - b < s_i < b$
ii) $r = \sum_{i\in I\!N} s_i \times b^{-i}$

e) in the continuous fraction representation a real number r is represented by a computable sequence of integers $s_0, \ldots, s_i \ldots$ such that

$$r = \lim_{i\to\infty} s_0 + \cfrac{1}{s_1 + \cfrac{1}{s_2 + \cfrac{1}{\ddots + \cfrac{1}{s_i}}}}$$

Representations a) and b) are used in [31] and in [5] respectively. Representations a) and b) are similar to the classical Cauchy sequence representation. Notice however that the constructive definition of a real number via a Cauchy sequence always requires the presence of a function defining the convergence rate. This convergence function can be the same for all Cauchy sequences, like in representation b), or can be specified for each Cauchy sequence, like in representation a). An informal justification for the necessity of introducing a function giving the convergence rate is the following: if the convergence rate of a Cauchy sequence $s_0, s_1, \ldots$ with limit x is not known then it is impossible to give any estimate of the value x after examining a finite subsequence $s_0, \ldots, s_i$, in fact any real number can be the limit of a Cauchy sequence starting with $s_0, \ldots, s_i$. This is of paramount importance in fact from a constructive point of view only finite part of an infinite sequence can be examined.

The representation c) is used in [6]. It can be considered a variant of the Cauchy sequence representation. Here a sequence of integers is used to describe a Cauchy sequence of rational p-adics numbers. A p-adic rational number is a

number that can be written in the form $m \times p^n$ with m and n integers. Note that just a subset of the Cauchy sequences of rationals can be described in this way. For the practical purposes representation c) is convenient: the algorithms for the arithmetic operations turn out to be simpler and more efficient when representation c) is used instead of representations a) or b).

The representation d) is similar to the standard digit representation. The main difference consists in introducing negative digits. This representation has been studied in [2], [6], [36].

The representation e) has been developed in [33] and is similar to the continuous fractions representation. The only difference is that in the standard continuous fraction notation only natural numbers are used. In this case it is necessary to use also negative integers.

The representations described above do not make explicit use of intervals. Anyway a perfectly equivalent representations, based on rational intervals can be given.

Let us consider for example representation b). A real number r is defined by a Cauchy sequence of rational numbers $s_0, \ldots, s_i, \ldots$. If we examine a finite part of the sequence s we can give an estimation of the value r. From the element s_i we know that the value of r lies in the interval: $[s_i - 1/i, s_i + 1/i]$. The same informations given by s can then be given by the sequence of rational intervals: $[s_1 - 1, s_1 + 1], \ldots, [s_i - 1/i, s_i + 1/i], \ldots$.

Analogous considerations can be done for the other representations.

As mentioned above the representations presented d) and in e) are modifications of the digit representation and of the continuous fraction representation respectively. The reason for these modifications is that the standard representations are not suitable for real number computation. Using the standard representations even the most fundamental functions such as addition or multiplication are not computable.

Here is a simple example that illustrates the inadequacy of the standard decimal representation. We show that *no algorithm can compute the multiplication by* 3. An hypothetical algorithm for this function will not be able to generate the first digit of the result when it receives as input the value 0.333... . In this case there are two possible results, namely 1.000... and 0.999... .

If the algorithm generates 1 as first digit, this happens after the algorithm has examined a finite number of digits of the argument. Let us suppose that the first n digits have been examined before generating 1. Then the algorithm generates 1 as first digit also when it receives as input the string $0.\overbrace{33\ldots3}^{n}20\ldots$. But this in incorrect, the exact result should then be $0.\overbrace{99\ldots9}^{n}6\ldots$.

An analogous consideration can be done if the algorithm generates 0 as first digit.

Similar examples show also that the other arithmetic operations are not computable.

It is clear that the problem presented above is not caused by the choice of the basis 10 for the representation of real numbers. The same problem would arise for any other basis.

The introduction of *negative digits* is a simple way to overcome these difficulties.

Going back to the previous example and we can easily show how the introduction of negative digits solves the difficulty.

The algorithm for the multiplication by 3 can in fact safely generate 1, as the first digit, after having read the first two digits of the string 0.333... . We can easily observe that if the input becomes $0.3(-9)(-9)... = 0.2$ the output can become $1.(-4)000... = 0.6$. If the input becomes $0.3999... = 0.4$ the output can become 1.2000... .

A perfectly similar consideration can be done for the continuous fraction representation.

In the following chapter we prove that all the previous real number representations are computationally equivalent, in the sense that they characterize the same class of computable reals and computable real functions. In constructive mathematics other representations of real numbers, not conputationally equivalent to the previous one, are considered, for example representations based on Dedekind cuts [31]. Anyway these representations are not suitable for a practical computation and they are not discussed in this thesis.

Chapter 3

The language PCFS

We define here a simple typed lambda calculus that we shall use to define computable real numbers and computable real functions.

This lambda calculus, called PCFS is for many aspect similar to the language PCF presented in [21]. The main difference between the two languages is that in PCFS there is a new unary type constructor "stream of".

We choose a typed lambda calculus as a formalism for expressing the computable functions because it is simple and thoroughly investigated. Moreover the lambda calculus is a paradigm for functional programming languages, and hence many problems typical of programming languages can be discussed in this setting.

In the following we present the language PCFS and we will give an operational semantics to it. Several different forms of real number representations will be "implemented" in PCFS. For each real number representation we define a full hierarchy of computable real functions.

3.1 The calculus

PCFS is a typed lambda calculus characterized by having a call-by-name strategy of evaluation and by containing in its set of types, together with every type, the type of streams of objects of that type. In this thesis streams will be used to represent real numbers.

3.1.1 Syntax

The set T of type expressions of $PCFS$ is defined by the grammar:

$$\tau := N \mid B \mid \tau \to \tau \mid S(\tau)$$

where τ is a metavariable ranging over the set of types, N and B are the type constants for naturals and booleans respectively. Types N and B are called

ground types. S is a type constructor, $S(\tau)$ is the type of streams of elements in τ.

Remark The grammar here defined does not contain the product type constructor. Functions of several arguments can be viewed as functions of a single argument by currying:
$(\sigma_1 \times \sigma_2) \to \sigma \ := \ \sigma_1 \to (\sigma_2 \to \sigma)$.

The set E of expressions of PCFS is defined by the grammar:

$$e \ := \ x^\tau \mid c^\tau \mid (e^{\tau \to \tau} e^\tau) \mid (\lambda x^\tau . e^\tau)$$

where x^τ is a metavariable over a countable set of variables Var_τ of type τ, and c^τ is a metavariable over the set of constants C. In the following we will use also the symbol $y, \alpha, \ldots$ as metavariable over set of the variables. When no confusion arises we will omit the indication of the type superscript τ in the terms e^τ.

The set $FV(e)$ of free variables over in e is defined by:
$FV(x) := \{x\}$, $FV(c) := \emptyset$, $FV((ee')) := FV(e) \cup FV(e')$, $FV((\lambda x.e)) := FV(e) - \{x\}$.

The application of terms (ee') is understood to be associative to the left. That is the term $e_1 e_2 \ldots e_n$ stands for $(\ldots (e_1 e_2) \ldots e_n)$

$e[e'/x]$ denotes the result of substituting the term e' in all the free occurrences of x in the term e, making appropriate renaming in the bound variables of e in order to avoid capturing.

The constants are:

$$k_0 \ \ldots \ k_n \ \ldots : N$$

$$tt, ff : B$$

$$\text{pred} \, , \text{succ} \ : N \to N, \qquad Z \ : N \to B$$

$$\text{cond}_\tau : B \to \tau \to \tau \to \tau,$$

$$\text{car}_\tau : S(\tau) \to \tau, \qquad \text{cdr}_\tau : S(\tau) \to S(\tau), \qquad \text{cons}_\tau : \tau \to S(\tau) \to S(\tau),$$

$$Y_\tau : (\tau \to \tau) \to \tau$$

Notice that there is no constant for the empty streams, the only stream constructor is *cons*. Closed expressions of stream type have to be constructed using the fix-point operator Y. As a consequence, the language does not contain any finite string. All the stream expressions denote infinite or diverging streams. We have designed the language in this way because only the infinite streams are meaningful in all the representations for real numbers that we consider.

Type assignments and type constraints are defined as usual.

3.1.2 Operational Semantics

The semantics is given by a *reduction relation (input-output)*, $\Rightarrow$ between closed terms.

If $e \Rightarrow v$ we say that v is the *value* of the term e. A term e *converges* ($e \downarrow$) if there is a term v such that $e \Rightarrow v$.

The definition of the relation $\Rightarrow$ is given by the following set of reduction rules

constant:

$$c \Rightarrow c \qquad \text{for all } c \in C$$

$$\operatorname{succ} k_i \Rightarrow k_{i+1}$$

$$\operatorname{pred} k_{i+1} \Rightarrow k_i$$

$$\operatorname{pred} k_0 \Rightarrow k_0$$

$$Z\ k_0 \Rightarrow tt$$

$$Z\ k_{i+1} \Rightarrow ff$$

stream:

$$\frac{e_1 \Rightarrow v}{\operatorname{car}_\tau(\operatorname{cons}_\tau e_1 e_2) \Rightarrow v}$$

$$\frac{e_2 \Rightarrow v}{\operatorname{cdr}_\tau(\operatorname{cons}_\tau\ e_1\ e_2) \Rightarrow v}$$

$$\operatorname{cons}_\tau\ e_1 \Rightarrow \operatorname{cons}_\tau\ e_1$$

$$\operatorname{cons}_\tau\ e_1\ e_2 \Rightarrow \operatorname{cons}_\tau\ e_1\ e_2$$

conditional:

$$\frac{e_0 \Rightarrow tt \qquad e_1 \Rightarrow v}{\operatorname{cond}_\tau\ e_0\ e_1\ e_2 \Rightarrow v}$$

$$\frac{e_0 \Rightarrow ff \qquad e_2 \Rightarrow v}{\operatorname{cond}_\tau\ e_0\ e_1\ e_2 \Rightarrow v}$$

$$\operatorname{cond}_\tau\ e_0 \Rightarrow \operatorname{cond}_\tau\ e_0$$

$$\operatorname{cond}_\tau\ e_0\ e_1 \Rightarrow \operatorname{cond}_\tau\ e_0\ e_1$$

fixed point:

$$\frac{e(Y_\tau\ e) \Rightarrow v}{Y_\tau\ e \Rightarrow v}$$

application:

$$\lambda x.e \Rightarrow \lambda x.e$$

$$\frac{e_1 \Rightarrow \lambda x.e_0 \qquad e_0[e_2/x] \Rightarrow v}{e_1\ e_2 \Rightarrow v}$$

It is easy to prove that for every term e there is at most one term v such that $e \Rightarrow v$, that is, $\Rightarrow$ is a partial functions between terms.

3.2 Real Number Computation in PCFS

In order to represent real numbers in PCFS it is necessary to represent integer and rational numbers in PCFS. This is done via an effective enumeration of integer and rational numbers.

An alternative solution can be that of extending PCFS with the product type constructor and to represent integers and rationals by pairs and triples of natural numbers. The two solutions are computationally equivalent. We choose the first one because it does not require the introduction of the product type constructor, and therefore it makes the description of PCFS simpler.

In the following π^2 is an effective enumeration of pairs of natural numbers. π^2 is given by two functions π^2_1 and π^2_2, $\pi^2(n) = \langle \pi^2_1(n), \pi^2_2(n) \rangle$. π^2 realizes the dove-tail enumeration of pairs, that is $\pi^2_1(n)$ and $\pi^2_2(n)$ are the only natural numbers satisfying the equation

$n = (\pi^2_1(n) + \pi^2_2(n)) \times (\pi^2_1(n) + \pi^2_2(n) + 1)/2 + \pi^2_2(n)$

more explicitly, if we indicate with $c(n)$ the integer part of $(-1+\sqrt{1+8n})/2$ then $\pi^2_2(n) = n - c(n) \times (c(n)+1)/2$ and $\pi^2_1(n) = c(n) - \pi^2_2(n)$

Effective enumerations $z : I\!N \to Z$ and $q : I\!N \to Q$ for the integer and rational numbers are then defined by:

$z(n) = \pi^2_1(n) - \pi^2_2(n)$

$q(n) = (\pi^2_1(\pi^2_1(n)) - \pi^2_2(\pi^2_1(n)))/\pi^2_2(n)$

Using the functions z and q it is then possible to use PCFS terms of type N to represent integers and rationals.

Two partial functions can be defined $[\![\ ]\!]_Z : E_N \to Z$ and $[\![\ ]\!]_Q : E_N \to Q$ are defined by:

$[\![e]\!]_Z = i$ if $e \Rightarrow k_n$ and $z(n) = i$

$[\![e]\!]_Q = r$ if $e \Rightarrow k_n$ and $q(n) = r$

These functions are partial because they are undefined on the divergent elements. The two partial functions associate to each PCFS term e^N the rational or the integer represented by it.

Partial functions, like $[\![\ ]\!]_Z$, generate expressions that can denote partial elements, that is elements that are not necessary defined. For example the elements $[\![Y_N \lambda x.x]\!]_Z$.

A remark about the equality relation is here necessary. We use two different equality symbols, "$=$" and "$\simeq$". With $t = s$ we denote that t and s are both defined elements and they are equal; with $t \simeq s$ we denote that if one of t and s is defined then so is the other and they are equal.

To represent real numbers inside PCFS it is sufficient to translate in PCFS any one of the different constructive representations for real number given in the previous chapter. Note that in each one of these definitions, a real number is represented by a sequence of (finite) elements. Inside PCFS there are two natural ways of representing sequences of elements of type τ: by using streams over τ or by using function from N to τ.

The Cauchy sequence representation in definition 1 b) is translated inside PCFS by any one of the two partial representation functions defined below:

Definition 3 *Two partial functions* $\mathbb{R}^1[\![\ ]\!] : E_{S(N)} \to \mathbb{R}$ *and* $\mathbb{R}^{1'}[\![\ ]\!] : E_{N\to N} \to \mathbb{R}$*, are defined by:*
$R^1[\![e]\!] = r$ *if there is a sequence of natural numbers* $l_0, l_1, \ldots$ *such that:*
i) $\forall n. car_\tau(cdr^n e) \Rightarrow k_{l_n}$
ii) $\mid q(l_n) - q(l_{n+1}) \mid \leq 2^{-n}$
iii) $r = \lim_{n\to\infty} q(l_n)$
$R^{1'}[\![e]\!] = r$ *if there is a sequence of natural numbers* $l_0, l_1, \ldots$ *such that:*
i) $\forall n. e\, k_n \Rightarrow k_{l_n}$
ii) $\mid q(l_n) - q(l_{n+1}) \mid \leq 2^{-n}$
iii) $r = \lim_{n\to\infty} q(l_n)$.

We called the functions in the form $R^i[\![\ ]\!]$ partial representation functions. Each partial representation function gives rise to a different definition of computable real number.

Definition 4 *Given a real partial representation functions* $R^i[\![\ ]\!]$ *a real number* r *is* language-computable *with respect to* $R^i[\![\ ]\!]$ *if there is a PCFS term* e *such that:*
$R^i[\![e]\!] = r$

It is easy to show that the two real partial representation functions generate equivalent definitions of computable real numbers. In fact there is an effective way to pass from the stream representation to the function on natural number representation. A general proof of this fact is given below.

Proposition 1 *For every type* τ *there are* 2 *PCFS terms s-fn and fn-s of type* $(N \to \tau) \to S(\tau)$ *and* $S(\tau) \to (N \to \tau)$ *respectively, such that:*
i) for every term l *with type* $S(\tau)$ *and for every natural number* n*:*
$car(cdr^i\ l) \Rightarrow v$ *iff* $s\text{-}fn\, l\, k_i \Rightarrow v$,
ii) for every term f *with type* $N \to \tau$ *and for every natural number* n*:*
$car(cdr^i(fn\text{-}s\ f)) \Rightarrow v$ *iff* $f\ k_i \Rightarrow v$.

Proof. It is not difficult to show that the following terms satisfy *i*) and *ii*).

$$\text{fn-s } = \lambda f. Y_{N\to S(\tau)}(\lambda L.\lambda n.\, \text{cons}(f\ n)(L(\text{succ}\, n)))k_0$$

$$\text{s-fn } = Y_{S(\tau)\to N\to\tau}(\lambda F.\lambda l'.\lambda n.\, \text{if} Z\ n\ \text{then}\, \text{car} l'\ \text{else}\, F(\text{cdr} l')(\text{pred}\, n))$$

□

Corollary 2 *The partial representation functions* $R^1[\![\ ]\!]$ *and* $R^{1'}[\![\ ]\!]$ *give rise to equivalent definitions of real computable number.*

We now translate inside PCFS the real representations presented in definition 2 c) and d). Also in these cases there is a choice between representing sequences by streams or by functions on natural numbers. Since it is possible to pass in an effective way from one representation to the other, this choice does not bring any significant difference from the computational point of view. We limit ourselves to consider the stream representation.

Definition 5 *Two real partial representation functions* $R^2[\![\ ]\!]$, $R^3[\![\ ]\!] : E_{S(N)} \to I\!R$ *are defined by:*

$R^2[\![e]\!] = r$ *if there is a sequence of natural numbers* $l_0, l_1, \ldots$ *such that:*
i) $\forall n. car(cdr^n e) \Rightarrow k_{l_n}$
ii) $\forall n. \mid 2 \times z(l_n) - z(l_{n+1}) \mid \leq 1$
iii) $r = \lim_{n\to\infty} z(l_n)/2^n$

$R^3[\![e]\!] = r$ *if there is a sequence of natural numbers* $l_0, l_1, \ldots$ *such that:*
i) $\forall n. car(cdr^n e) \Rightarrow k_{l_n}$
ii) $\forall n \geq 1\ l_n \in \{0, 1, 2\}$
iii) $r = z(l_0) + \sum_{n\in N}(l_n - 1)/2^n$

We could introduce and discuss yet another partial representation function correspondig to the definition of a real nunber as a continuous fraction. Although this representation would be not substantiall more difficult to analyze than those in the definition above, it would require a large number of tedious technicalities.

We prove now that it is possible to pass in an effective way from one representation to the other. In fact:

Proposition 3 *There are three PCFS terms* $conv_{2-1}$, $conv_{3-2}$ *and* $conv_{1-3}$ *such that for every term* $e \in E_{S(N)}$ *the following equalities hold:*

$R^1[\![e]\!] \simeq R^2[\![conv_{2-1}\ e]\!]$
$R^2[\![e]\!] \simeq R^3[\![conv_{3-2}\ e]\!]$
$R^3[\![e]\!] \simeq R^1[\![conv_{1-3}\ e]\!]$

Proof. As mentioned above each partial representation function R^i subsumes a representation of the real numbers by sequences of (rational or integer) numbers. We first show that it is possible to translate, in an effective way, one sequence representation into another.

Given a Cauchy sequence of rationals $s_0, s_1, \ldots$ representing a real number r and such that $\mid s_{i+1} - s_i \mid \leq 2^{-i}$ then the sequence of intergers
$\text{round}(s_1 \times 2), \ldots, \text{round}(s_i \times 2^i) \ldots$ defines the same real r through the representation subsumed by the function $R^2[\![\ ]\!]$. (with $\text{round}(x)$ we denote the integer approximation of the rational number x).

Similarly, given a sequence of integer $s_0, s_1, \ldots$ denoting a real r through the representation subsumed by the function R^2 then the sequence
$s_0, (s_1 - 2 \times s_0), \ldots, (s_{i+1} - 2 \times s_i), \ldots$
is a negative digit representation of r.

Finally given a negative digit representation $s_0, s_1, \ldots$ of r, the sequence
s_0, $(s_0 + s_1 \div 2), \ldots, (\sum_{j=0}^{i} s_j \times 2^{-i}), \ldots$
is a Cauchy sequence representing r.

The terms conv_{i-j} are just the implementations, inside, PCFS of the above translations.

As an example we present here the definition of conv_{2-1} and of conv_{3-2}.

Before giving the two terms it is necessary to introduce other terms defining several auxiliary functions.

It is straightforward to prove that the arithmetic functions on naturals, integers and rationals can be implemented in PCFS.

That is, there are terms:

plus_N, times_N, minus_N, div_N, plus_Z, times_Z, minus_Z, div_Z, plus_Q, times_Q, minus_Q and div_Q,

such that appropriate equations of form:

$[\![\mathrm{plus}_Z\ e\ e']\!]_Z \simeq [\![e]\!]_Z + [\![e']\!]_Z$, $[\![\mathrm{times}_Z\ e\ e']\!]_Z \simeq [\![e]\!]_Z \times [\![e']\!]_Z$ ...

are satisfied

Other terms needed in the proof are:
conv_{N-Z}, conv_{Z-Q}, abs, and round.

These terms satisfy following equalities:

$[\![(\mathrm{conv}_{N-Z}\ e)]\!]_Z \simeq [\![e]\!]_N$

$[\![(\mathrm{conv}_{N-Q}\ e)]\!]_Q \simeq [\![e]\!]_Z$

$[\![(\mathrm{abs}\ e)]\!]_N \simeq \mid [\![e]\!]_Z \mid$

$[\![(\mathrm{round}\ e)]\!]_Z \simeq$ the approximate value of $[\![e]\!]_Q$

We necessitate also of a set of terms $\mathrm{map}_{\tau\to\tau'}$ of type $(\tau \to \tau') \to S(\tau) \to S(\tau')$. Given a function $f : \tau \to \tau'$ and a stream α of type $S(\tau)$, $(\mathrm{map}_{\tau\to\tau'} f\alpha)$ is the stream obtained applying to each element of α the function f.

$$\mathrm{map}_{\tau\to\tau'} := \lambda f.Y_{S(\tau)\to S(\tau')}(\lambda F.\lambda\alpha.\mathrm{cons}(f(\mathrm{car}\ \alpha))(F(\mathrm{cdr}\ \alpha))))$$

Finally we can give the definitions:

$$\mathrm{conv}_{2-1} := Y(\lambda F.\lambda\alpha.\mathrm{cons}\ \ (\mathrm{round}(\mathrm{car}(\mathrm{cdr}(\alpha))))\ (F(\mathrm{map}\ (\lambda n.(\mathrm{times}(\mathrm{conv}_{N-Q}k_2)n)(\mathrm{cdr}\alpha))))$$

$$\mathrm{aux\text{-}conv}_{3-2} := Y(\lambda F.\lambda\alpha.\mathrm{cons}\ (\mathrm{abs}((\mathrm{plus}(\mathrm{conv}N - Zk_1)(\mathrm{minus}(\mathrm{car}(\mathrm{cdr}\alpha))(\mathrm{time}(\mathrm{conv}N - Zk_2)(\mathrm{car}\alpha)))))(F(\mathrm{cdr}\alpha))))$$

$$\mathrm{conv}_{2-3} := \lambda\alpha.\mathrm{cons}(\mathrm{car}\alpha)(\mathrm{aux\text{-}conv}_{2-3}\alpha)$$

□

By definition 4 each partial representation function gives rise to a different definition of computable real number.

We can now state the theorem.

Theorem 4 *The partial representation functions $R^1[\![\ ]\!]$, $R^{1'}[\![\ ]\!]$, $R^2[\![\ ]\!]$ and $R^3[\![\ ]\!]$ generate equivalent definitions for computable real.*

Proof. The theorem is an immediate consequence of Proposition 3.

Notation. We indicate with $\mathbb{R}^l$ the set of the computable real numbers.

3.3 Computable Functions.

Here we define the notion of unary computable functions on real numbers. In the following the definition will be extended to higher order functions.

In this chapter we consider just functions defined over the set of computable reals. Functions defined over the whole real line will be considered in the following chapters.

There is a uniform way to pass from a partial representation function for the real numbers to a partial representation function for the functions on computable real numbers.

Definition 6 *Given a partial representation function $R^i[\![\ ]\!] : E_\tau \to \mathbb{R}$ for the real numbers, a partial representation function for the set of functions $R^i[\![\ ]\!]_{r\to r} : E_{\tau\to\tau} \to \mathbb{R}^l \to \mathbb{R}^l$ is defined by:*
$R^i[\![e]\!]_{r\to r} = f$ *iff*
$\forall x \in \mathbb{R}^l.\forall e' \in E_\tau.R^i[\![e']\!] = x \ \Rightarrow\ R^i[\![e(e')]\!] = f(x)$.

Each partial representation function gives rise to a different definition of computable real functions.

Definition 7 *A function $f : \mathbb{R}^l \to \mathbb{R}^l$ is computable with respect to the real partial representation function $R^i[\![\ ]\!]$ if there is a PCFS term e such that:*
$R^i[\![e]\!] = f$.

Theorem 5 *The real partial representation functions introduced in definitions 3 and 5 give rise to equivalent definitions of real computable functions.*

Proof. Again the proof is an immediate consequence of proposition 3.

It is possible to extend the definitions of partial representation function and of computability to arbitrary higher order functions.

The set T' of types on reals considered is generated by the grammar:
$\sigma = r \mid \sigma \to \sigma$
where r is the only type constant whose intended meaning is the type of reals.

Definition 8 *Given a partial representation function $R^i[\![\ ]\!] : E_\tau \to R$ we define by structural induction on the type $\sigma \in T'$ the function $t^i : T' \to T$, the families of sets $\mathbb{R}^{aux-i}_\sigma$, $\mathbb{R}^{l-i}_\sigma$ and the family of functions $R^i[\![\ ]\!] : E_{t^i(\sigma)} \to \mathbb{R}^{aux-i}_\sigma$ by:*
$t(r)^i = \tau$
$t^i(\sigma \to \sigma') = t^i(\sigma) \to t^i(\sigma')$

$\mathbb{R}_r^{aux-i} := \mathbb{R}$
$\mathbb{R}_{\sigma\to\sigma'}^{aux-i} := \mathbb{R}_\sigma^{l-i} \to \mathbb{R}_{\sigma'}^{l-i}$
$R^i[\![\ ]\!]_r := R^i[\![\ ]\!]$
$R^i[\![e]\!]_{\sigma\to\sigma'} = f$ *iff* $\forall x \in \mathbb{R}_\sigma^{l-i}.\forall e' \in E_{t^i(\sigma)}.R^i[\![e']\!]_\sigma = x \ \Rightarrow\ R^i[\![e'(e)]\!]_{\sigma'} = f(x)$
$\mathbb{R}_\sigma^{l-i} = \{x \in \mathbb{R}_\sigma^{aux-i} \mid \exists e \in E_{t^i(\sigma)}.x = R^i[\![e]\!]_\sigma\}$

We call the sets $\mathbb{R}_\sigma^{l-i}$ the sets of *language-computable* functions on real number relatively to the real representation i.

The following theorem generalizes the equivalence results between the different notions of computability.

Proposition 6 *Given two partial representation functions for the real numbers $R^i[\![\ ]\!] : E_\tau \to \mathbb{R}$ and $R^{i'}[\![\ ]\!] : E_{\tau'} \to \mathbb{R}$, if there is an effective way to pass from one representation to the other, that is if there are two term conv and conv$'$ such that*

1) $\forall e \in E_\tau R^i[\![e]\!] \ \simeq\ R^{i'}[\![conv(e)]\!]$

2) $\forall e \in E_{\tau'} R^{i'}[\![e]\!] \ \simeq\ R^i[\![conv'(e)]\!]$

then for every type $\sigma \in T'$ the sets $\mathbb{R}_\sigma^{l-i}$ and $\mathbb{R}_\sigma^{l-i'}$ coincide.

Proof. We first extend the terms conv and conv$'$ to a set of terms for the conversion of terms of an arbitrary type $\sigma \in T'$.
$\mathrm{conv}_r := \mathrm{conv}$
$\mathrm{conv}'_r := \mathrm{conv}'$
$\mathrm{conv}_{\sigma\to\sigma'} := \lambda f.\mathrm{conv}_{\sigma'} \circ f \circ \mathrm{conv}'_\sigma$
$\mathrm{conv}'_{\sigma\to\sigma'} := \lambda f.\mathrm{conv}'_{\sigma'} \circ f \circ \mathrm{conv}_\sigma$
It is now immediate to prove that:
$\forall\sigma \in T' \forall e \in E_{t^i(\sigma)} R^i[\![e]\!]_\sigma \ \simeq\ R^{i'}[\![\mathrm{conv}_\sigma(e)]\!]_\sigma$
$\forall\sigma \in T' \forall e \in E_{t^{i'}(\sigma)} R^{i'}[\![e]\!]_\sigma \ \simeq\ R^{i'}[\![\mathrm{conv}'_\sigma(e)]\!]_\sigma$
The thesis now follows directly.

□

An immediate consequence is that all the real number representations presented so far, give rise to the same notion of computable real and computable real function. We can therefore use the notion of language-computable function over real numbers without mentioning the real representation used. For every $\sigma \in T'$ the sets $\mathbb{R}_\sigma^{l-i}$ ($i \in 1, 1', 2, 3$) are equal no matter which i is considered. Therefore the reference to the real representation is suppressed and the set of computable functions will be indicated just with $\mathbb{R}_\sigma^l$.

As mentioned above the computable functions over the reals presented in this chapter have a domain restricted to computable reals. A definition of computable function over the whole real line can be obtained introducing the notion of "oracle term" that is extending the language with an infinite set of constants $\{e_x \mid x \in \mathbb{R}\}$ having the property that $R^i[\![e_x]\!] = r$. Anyway we do not follows this approach. Instead in the next chapter we show how to define a notion of computability for the functions defined over the whole real line using domain theory.

Chapter 4

A Domain Theoretic Approach to Computable Reals

In this section we give a second definition of computability on real numbers. In brief the method we use is the following. We start by giving a denotational semantics to the language PCFS. Then, using the classical notion of computability on domains, we define a new notion of computability on real numbers. This new notion of computability is conceptually different from the one of language-computability introduced in the previous chapter. We will show however how to establish a strict relation between these two notions.

The denotational semantics is given using ω-algebraic consistently complete cpo's, i.e. Scott-domains.

For completeness we briefly summarize the basic definitions of domain theory. Given a partial order $(D, \sqsubseteq)$ the following notation is used. $d \sqcup d'$ and $d \sqcap d'$ denote the least upper bound and the greatest lower bound of the elements $d, d' \in D$ respectively, if they exists. Given a subset S of D, $\bigsqcup S$ and $\sqcap S$ denote respectively the least upper bound, greatest lower bound of S, if they exists.

A *directed* subset of partial order $(D, \sqsubseteq)$ is a non empty subset S of D such that every pair of elements of S has an upper bound in S. A subset S of a partial order $(D, \sqsubseteq)$ is consistent, written $S \uparrow$, if it has an upper bound. $a \uparrow b$ denotes that the set $\{a, b\}$ is consistent.

A *complete partial order* (cpo) is a partial order $(D, \sqsubseteq)$ which has a least element $\bot_D$ and all the least upper bounds of directed subsets.

A *finite* element of a cpo $(D, \sqsubseteq)$ is an element $d \in D$ such that for any directed subset of S of D if $d \sqsubseteq \sqcup S$ then there is an element $s \in S$ such that $d \sqsubseteq s$. The set of finite elements of $(D, \sqsubseteq)$ is denoted with D°.

A cpo $(D, \sqsubseteq)$ is *algebraic* if, for each $d \in D$ the set $\{s \mid s \sqsubseteq d, s \in D^\circ\}$ is directed and $d = \sqcup\{s \mid s \sqsubseteq d, s \in D^\circ\}$.

A cpo $(D, \sqsubseteq)$ is ω-*algebraic* if it is algebraic and the set D° of finite elements is countable.

A cpo $(D, \sqsubseteq)$ is *consistently complete* if whenever two elements d, d' in D have an upper bounds they have a least upper bound. A *Scott domain* is a consistently complete ω-algebraic cpo.

A function $f : D \to D'$ from a cpo D to a cpo D' is *continuous* if for all S directed subset of D the following equality holds:
$f(\bigsqcup S) = \bigsqcup_{d \in S} f(d)$.
The set $[D \to D']$ of all the continuous functions from D to D' is itself a cpo under the pointwise ordering induced by the ordering on D'. If d, d' are finite elements of D and D' we denote with $(d \Rightarrow d')$ the member of $[D \to D']$ defined by

$$(d \Rightarrow d')(x) = \begin{cases} d' & \text{if } d \sqsubseteq x \\ \bot & \text{otherwise} \end{cases}$$

A function of the form $(d \Rightarrow d')$ is called *step function*.

If D and D' are Scott-domains so is $[D \to D']$. Its finite elements are the lub's of finite sets of step functions.

Let D be a cpo. Its lifting $D_\bot$ is the set $\{\langle 1, d\rangle \mid d \in D\} \cup \{\bot\}$ with the ordering
$x \sqsubseteq y$ if $x = \bot$ or $x = \langle 1, x'\rangle$, $y = \langle 1, y'\rangle$ and $x' \sqsubseteq y'$.
Clearly if D is a Scott-domain then $D_\bot$ is a Scott-domain.

Scott domains can be endowed with a topological structure called Scott topology.

In a Scott-domain D the *Scott topology* is the topology whose open sets are union of sets in the form $\overset{\vee}{d^\circ} = \{d \mid d^\circ \sqsubseteq d\}$ with $d^\circ \in D^\circ$

An *effective Scott domain* is a triple $(D, \sqsubseteq, en)$ such that $(D, \sqsubseteq)$ is a Scott domain and en is an enumeration of the finite elements D° such that the following relations are decidable:
i) $en(n) \uparrow en(m)$
ii) $en(n) = en(m) \sqcup en(m')$

An element d in a effective Scott domain $(D, \sqsubseteq, en)$ is *computable* if the set $\{n \mid en(n) \sqsubseteq d\} \subseteq I\!N$ is recursively enumerable.

4.1 Denotational Semantics for PCFS.

A collection of effective Scott domains $\{D_\tau\}$, one for each type is defined by induction on the type $\tau \in T$.

D_B and D_N are the standard flat domains of truthvalues and natural numbers: $D_B := \{\bot_B, tt, ff\}$ $D_N := \{\bot_N, 0, 1, 2, \ldots\}$

$D_{\tau \to \tau'} := [D_\tau \to D_{\tau'}]_\bot$ is the lifting of the domain of continuous functions from D_τ to $D_{\tau'}$

$D_{S(\tau)}$ is the domain composed by the (finite and infinite) strings of elements of D_τ, including the empty string. The order on $D_{S(\tau)}$ is defined by:
$[d_0.d_1.\ldots.d_i] \sqsubseteq [d'_0.d'_1.\ldots.d_j]$ se $i \leq j$ and for all $h\ 0 \leq h \leq i.\ d_h \leq d'_h$
$[d_0.d_1.\ldots.d_i] \sqsubseteq [d'_0.d'_1.d'_2.\ldots]$ if for all $h\ 0 \leq h \leq i.\ d_h \leq d'_h$
$[d_0.d_1.d_2.\ldots] \sqsubseteq [d'_0.d'_1.d'_2.\ldots]$ if for all $h \in I\!N\ .\ d_h \leq d'_h$

An alternative definition for $D_{S(\tau)}$ is: $D_{S(\tau)}$ is a minimal solution of the domain equation $D_{S(\tau)} \cong (D_{S(\tau)} \times D_\tau)_\perp$

The semantics interpretation function E has the form:

$$E : L \to Env \to \bigcup\{D_\tau\}$$

where Env is the set of environments. Environments ranged over by ρ are functions from Var to $\bigcup\{D_\tau\}$ respecting the type constraints, i.e.:

$$\rho(x^\tau) \in D_\tau$$

We denote with $\perp$ the environment that maps every variable to the bottom element of the corresponding domain. The context will clarify ambiguities in the use of $\perp$.

The definition of E is given by structural induction,

$$\begin{array}{lcl}
E[\![c]\!]_\rho & := & B[\![c]\!] \\
E[\![x^\tau]\!]_\rho & := & \rho(x^\tau) \\
E[\![(e^{\tau\to\tau'} e^\tau)]\!]_\rho & := & \mathrm{down}_{\tau\to\tau'}(E[\![e^{\tau\to\tau'}]\!]_\rho)(E[\![e^\tau]_\rho) \\
E[\![(\lambda x^\tau.e^{\tau'})]\!]_\rho & := & \mathrm{up}_{\tau\to\tau'}(\lambda a \in D_\tau.E[\![e^\tau]\!]_{(\rho[a/x])})
\end{array}$$

Where "down_τ" : $(D_\tau)_\perp \to D_\tau$ and "up_τ" : $D_\tau \to (D_\tau)_\perp$ are the obvious morphisms.

We define next the function B of constants interpretation.

$$B[\![k_n]\!] = n$$

$$\mathrm{down}_{N\to N}(B[\![\mathrm{succ}\,]\!])n = \begin{cases} n+1 & \text{if } n \leq 0 \\ \perp & \text{if } n = \perp \end{cases}$$

$$\mathrm{down}_{N\to N}(B[\![\mathrm{pred}\,]\!])n = \begin{cases} n-1 & \text{if } n > 0 \\ \perp & \text{if } n = \perp, 0 \end{cases}$$

$$\mathrm{down}_{N\to N}(B[\![Z\,]\!])n = \begin{cases} tt & \text{if } n > 0 \\ ff & \text{if } n = 0 \\ \perp & \text{if } n = \perp \end{cases}$$

$$\mathrm{down}_{\tau\to\tau}(\mathrm{down}_{\tau\to\tau\to\tau}(\mathrm{down}_{B\to\tau\to\tau\to\tau}(B[\![\mathrm{cond}_{B\to\tau\to\tau\to\tau}]\!])b)x)y := \begin{cases} x & if \quad b = tt \\ y & if \quad b = ff \\ \perp & if \quad b = \perp \end{cases}$$

$$\mathrm{down}_{\tau\to S(\tau)\to S(\tau)}(\mathrm{down}_{S(\tau)\to S(\tau)}(B[\![\mathrm{cons}_\tau]\!])a)\,[a_0.a_1.\ldots.] := [a.a_0.a_1.\ldots.]$$

$$\mathrm{down}_{S(\tau)\to\tau}(B[\![\mathrm{car}_\tau]\!])s := \begin{cases} a_0 & if \quad s = [a_0.a_1\ldots.] \\ \bot & if \quad s = [\,] \end{cases}$$

$$\mathrm{down}_{S(\tau)\to S(\tau)}(B[\![\mathrm{cdr}_\tau]\!])s := \begin{cases} [a_1\ldots.] & if \quad s = [a_0.a_1\ldots.] \\ [\,] & if \quad s = [\,] \end{cases}$$

$$B[\![Y_\tau]\!]) := \mathrm{up}(\bigsqcup_{n\in N}\{\lambda f.f^n\bot_\tau\})$$

4.2 Relation between denotation and operational semantics

The value denoted by a term determines its operational behavior, as will be made clear by the following proposition.

Proposition 7 *For every closed term e the following facts hold*
i) if $e \Rightarrow v$ *then* $E[\![e]\!]_\bot = E[\![v]\!]_\bot$.
ii) $e\downarrow$ *if and only if* $E[\![e]\!]_\bot \neq \bot$

Proof.
i) can be proved straightforward. It is sufficient to observe that every evaluation rule preserves the denotational equality. That is for every evaluation rule:

$$\frac{e_0 \Rightarrow v_0 \quad \ldots \quad e_i \Rightarrow v_i}{e \Rightarrow v}$$

if for every index h, $0 \leq h \leq i$. $[\![e_h]\!]_\bot = E[\![v_h]\!]_\bot$ then $E[\![e]\!]_\bot = E[\![v]\!]_\bot$

As far as ii) is concerned first observe that for every value term v $E[\![v]\!]_\bot \neq \bot$. It follows that if $e \Rightarrow v$ then $E[\![e]\!]_\bot \neq \bot$.

Implication ii) can then be proved by structural induction, on the structure of the term e. To this aim it is necessary to carefully define the induction hypothesis. We use a generalization of Tait's computability technique used also by Plotkin in [21]. A set of predicates Comp_τ over PCFS terms is defined by induction on the types by:
i) a closed term e^τ, with τ ground type satisfies property Comp_τ if $e\downarrow \Leftrightarrow E[\![e]\!]_\bot \neq \bot$
ii) a closed term $e^{\tau\to\tau'}$, satisfies property $\mathrm{Comp}_{\tau\to\tau'}$ if $e\downarrow \Leftrightarrow E[\![e]\!]_\bot \neq \bot$ and whenever e'^τ is a closed term satisfying the property Comp_τ then $e(e')$ satisfies the property Comp_τ.
iii) a closed term $e^{S(\tau)}$, satisfies property $\mathrm{Comp}_{S(\tau)}$ if $e\downarrow \Leftrightarrow E[\![e]\!]_\bot \neq \bot$ and for every natural number n $\mathrm{car}(\mathrm{cdr}^n e)$ satisfies the property Comp_τ.
A open term e^τ with free variables $x_1, x_2, \ldots x_i$ satisfies the property Comp_τ if for every $e_1, \ldots e_i$ closed terms satisfying the properties Comp, $e[e_1/x_1]\ldots[e_i/x_i]$ satisfies the property Comp.

As is standard using this technique we call *computable* a term satisfying property Comp.

Proposition 7 can be easily derived once it is proved that every term is computable.

First we need to show that computability satisfies the following:

Lemma 8 *For every pair of PCFS closed terms e^τ and e'^τ such that $e \Rightarrow v$ iff $e' \Rightarrow v$, and $E[\![e]\!]_\perp = E[\![e']\!]_\perp$ then e is computable if and only if e' is computable.*

The proof is a straightforward inductions on the type τ.

Lemma 9 *Every term is computable.*

The proof is by induction of the structure of terms.

Clearly every variable is computable.

It is straightforward to prove that every constant other than Y_τ is computable. The constants Y_τ will be considered at the end of the proof.

As an example we give here the details of the proof that $\mathrm{cons}^{\tau\to S(\tau)\to S(\tau)}$ is computable. It is necessary to prove that if e^τ and $e'^{S(\tau)}$ are computable terms then $\mathrm{cons}\, e\, e'$ is computable.

In this case the only difficulty is to prove that for every natural number n $\mathrm{car}(\mathrm{cdr}^n(\mathrm{cons}\, e\, e'))$ is computable.

For $n = 0$ we have that:
$\mathrm{car}(\mathrm{cons}\, e\, e') \Rightarrow v$ iff $e \Rightarrow v$, $\qquad E[\![\mathrm{car}(\mathrm{cons}\, e\, e')]\!]_\perp = E[\![e]\!]_\perp$
and the computability of $\mathrm{car}(\mathrm{cons}\, e\, e')$ follows from the computability of e and Lemma 8,
the case $n > 0$ are analogous:
$\mathrm{car}(\mathrm{cdr}^n(\mathrm{cons}\, e\, e')) \Rightarrow v$ iff $\mathrm{car}(\mathrm{cdr}^{n-1}e') \Rightarrow v$ and
$E[\![\mathrm{car}(\mathrm{cdr}^n(\mathrm{cons}\, e\, e'))]\!]_\perp = E[\![\mathrm{car}(\mathrm{cdr}^{n-1}e')]\!]_\perp$
and the computability of $\mathrm{car}(\mathrm{cdr}^n(\mathrm{cons}\, e\, e'))$ follows from computability of e' and Lemma 8.

Next we prove that the application $e\, e'$ of two computable terms e, e' is computable. This is an easy consequence of the fact that if $x_1, \ldots, x_n$ are the free variables contained in $e\, e'$ then
$(e\, e')[e_1/x_1] \ldots [e_n/x_n] = (e[e_1/x_1] \ldots [e_n/x_n])(e'[e_1/x_1] \ldots [e_n/x_n])$

To prove the computability of $\lambda x.e$ it is sufficient to prove that if $x_1, \ldots, x_n$ are the free variables of $\lambda x.e$ and $e', e_1, \ldots e_n$ are closed computable terms then $((\lambda x.e)[e_1/x_1] \ldots [e_n/x_n])e'$ is computable. But
$((\lambda x.e)[e_1/x_1] \ldots [e_n/x_n])e' \Rightarrow v$ iff $e[e_1/x_1] \ldots [e_n/x_n][e'/x] \Rightarrow v$ and
$E[\![((\lambda x.e)[e_1/x_1] \ldots [e_n/x_n])e']\!]_\perp = E[\![e[e_1/x_1] \ldots [e_n/x_n][e'/x]]\!]_\perp$
again the computability of $\lambda x.e$ is a consequence of Lemma 8.

It remains to prove that the constants Y_τ are computable,

Some preliminary steps are necessary:

We define the following terms $\Omega_\tau := Y_\tau(\lambda x^\tau.x^\tau)$, $Y_\tau^0 := \lambda f^{\tau\to\tau}.\Omega_\tau$,
$Y_\tau^{i+1} := \lambda f^{\tau\to\tau}.f^{\tau\to\tau}(Y_\tau^i f^{\tau\to\tau})$.

Since $E[\![\Omega_\tau]\!]_\perp = \perp$, Ω_τ is trivially computable.

By induction it is possible to derive that also the terms Y_τ^i are computable.

It is easy to prove that $E[\![Y_\tau]\!]_\perp = \bigsqcup_{i\in I\!N} E[\![Y_\tau^i]\!]_\perp$
and by continuity: $E[\![e[Y_\tau/x]]\!]_\perp \neq \perp$ iff there is an i such that:
$E[\![e[Y_\tau^i/x]]\!]_\perp \neq \perp$

Moreover it is easy to prove that if $e[Y_\tau^i/x] \downarrow$ then $e[Y_\tau/x] \downarrow$.

Observe that every type τ can be written in the form:
$\tau_{1,1} \to \tau_{1,2} \to \ldots \tau_{1,i_1} \to S(\tau_{2,1} \to \ldots \tau_{2,i_2} \to S(\ldots S(\tau_{j,1} \to \ldots \tau_{j,i_j}) \ldots)$
with τ_{j,i_j} ground type. Note that the case when $i_1 = 0$ or $j = 1$ are particular cases of this schema.
It is easy to check that the following characterization of the property Comp_τ is valid: a closed term e^τ, with
$\tau = \tau_{1,1} \to \tau_{1,2} \to \ldots \tau_{1,i_1} \to S(\tau_{2,1} \to \ldots \tau_{2,i_2} \to S(\ldots S(\tau_{j,1} \to \ldots \tau_{j,i_j}) \ldots)$,
is computable if and only for every n-tuple of computable terms
$e_{1,1}, \ldots, e_{1,i_1}, \ldots, e_{j,i_j-1}$ and for every n-tuple of natural numbers $n_1, \ldots n_j$, all the terms:
$(e\ e_{1,1})$, $(e\ e_{1,1}\ e_{1,2})$, $\ldots$ $(\mathrm{car}(\mathrm{cdr}^{n_1}(e\ e_{1,1} \ldots e_{1,i_1})))$,
$\ldots$
$(\mathrm{car}(\mathrm{cdr}^{n_j}(\ldots(\mathrm{car}(\mathrm{cdr}^{n_1}(e\ e_{1,1} \ldots e_{1,i_1})))e_{2,1} \ldots e_{j,i_j-1})$
are such that their computation converges if and only if their denotation is different from $\perp$.

Using the above characterization of computability we can finally prove the computability of Y_τ.

Let $e_{1,1}, \ldots, e_{1,i_1}, \ldots, e_{h,1}, \ldots, e_{h,l}$ be a tuple of computable terms and $n_1 \ldots n_h$ a tuple of natural numbers,

then $E[\![(\mathrm{car}(\mathrm{cdr}^{n_h}(\ldots(\mathrm{car}(\mathrm{cdr}^{n_1}(Y_\tau e_{1,1} \ldots e_{1,i_1})))e_{2,1} \ldots))) \ldots e_{h,l}]\!]_\perp \neq \perp$ then there is an i such that

$E[\![(\mathrm{car}(\mathrm{cdr}^{n-h}(\ldots(\mathrm{car}(\mathrm{cdr}^{n_1}(Y_\tau^i e_{1,1} \ldots e_{1,i_1})))e_{2,1} \ldots))) \ldots e_{h,l}]\!]_\perp \neq \perp$

by computability of Y_τ^i,

$(\mathrm{car}(\mathrm{cdr}^{n-h}(\ldots(\mathrm{car}(\mathrm{cdr}^{n_1}(Y_\tau^i e_{1,1} \ldots e_{1,i_1})))e_{2,1} \ldots))) \ldots e_{h,l} \downarrow$

and this implies that:

$(\mathrm{car}(\mathrm{cdr}^{n-h}(\ldots(\mathrm{car}(\mathrm{cdr}^{n_1}(Y_\tau e_{1,1} \ldots e_{1,i_1})))e_{2,1} \ldots))) \ldots e_{h,l} \downarrow$

and from this the computability of Y_τ.

□

4.3 From Domains to Real Numbers.

Using domain theory we give now new definitions for computability on real numbers.

In the previous chapter we considered three different representations for real numbers. Each of them has been used to give a definition of language-computability. Similarly we use the same three representations to define a notion of computability on reals based on domain theory. We introduce therefore

three partial functions from $D_{S(N)}$ to $\mathbb{R}$. These partial functions are the domain theoretic analogues corespondents of the partial interpretation functions introduced in definitions 3 and 5

Notations. In all the representation used so far the real numbers are represented by strings of naturals, we will often write D_r to denote the domain $D_{S(N)}$. And more generally, where no confusion arises, we abbreviate the type $S(N)$ with the symbol r.

Definition 9 *The partial functions* $va^i : D_r \to \mathbb{R}$, $i \in \{1,2,3\}$ *are defined by:*

$va^1(d) := x$ *iff*
i) $d = [l_0.l_1....]$ *with* $l_n \neq \perp$
ii) $\mid q(l_n) - q(l_{n+1}) \mid \leq 2^{-n}$
iii) $x = \lim_{n\to\infty} q(l_n)$

$va^2(d) := x$ *iff*
i) $d = [l_0.l_1....]$ *with* $l_n \neq \perp$
ii) $\forall n. \mid 2 \times z(l_n) - z(l_{n+1}) \mid \leq 1$
iii) $x = \lim_{n\to\infty} z(l_n)/2^n$

$va^3(d) = x$ *iff*
i) $d = [l_0.l_1....]$ *with* $l_n \neq \perp$
ii) $\forall n \geq 1.\ l_n \in \{0,1,2\}$
iii) $x = z(l_0) + \sum_{n\in N}(l_n - 1)/2^n$

where z *and* q *are the enumeration functions of integers and rationals defined in the previous chapter.*

Proposition 10 *For every index* $i \in \{1,2,3\}$ *for every closed term* e *with type* $S(N)$
$R^i[\![e]\!] \simeq va^i(E[\![e]\!]_\perp)$

The proposition is a straightforward consequence of proposition 7.

Functions va^i can be extended to function spaces. The method used in defining the extension is similar to the one presented in the previous chapter for the function $R^i[\![\ ]\!]$.

Definition 10 *For every index* $i \in \{1,2,3\}$ *given an partial embedding function* $va^i : D_r \to \mathbb{R}$, *we define by structural induction on the type* $\sigma \in T'$ *the families of sets* $\mathbb{R}^{d-i}_\sigma$, $\mathbb{R}^{ad-i}_\sigma$, $D^{\dagger i}_\sigma$ *and the family of partial functions*
$va^i_\sigma : D_\sigma \to \mathbb{R}^{ad-i}_\sigma$

$\mathbb{R}^{d-i}_r := \mathbb{R}^{ad-i}_r := \mathbb{R}$

$\mathbb{R}^{ad-i}_{\sigma\to\sigma'} := \mathbb{R}^{d-i}_{\sigma} \to \mathbb{R}^{d-i}_{\sigma'}$

$va^i_r := va^i$
$va^i_{\sigma\to\sigma'}(d) := f \;\; \textit{iff} \;\; \forall x \in \mathbb{R}^{d-i}_{\sigma}.\, \forall d' \in D_\sigma.\, va^i_\sigma(d') = x \;\Rightarrow\; va^i_{\sigma'}(d(d')) = f(x)$

$\mathbb{R}^{d-i}_{\sigma} :=$ *the codomain of the function* va^i_σ

$D^{\dagger i}_\sigma := \{d \in D^i_\sigma \mid va^i_\sigma(d) \textit{ is defined }\}$

Functions contained in the sets $\mathbb{R}^{d-i}_{\sigma}$ are called constructive real functions.
The computable elements in $\mathbb{R}^{d-i}_{\sigma}$ are defined by:

Definition 11 *An element $f \in \mathbb{R}^{d-i}_{\sigma}$ is called* domain-computable *if there is a computable element $d \in D_\sigma$ such that: $f = va^i_\sigma(d)$.*

Given a type σ the sets $\mathbb{R}^{d-1}_{\sigma}, \ldots, \mathbb{R}^{d-3}_{\sigma}$ are equal and contain the same subset of computable elements. To prove this fact if is sufficient to show that it is possible to go effectively from one representation to the other.
We start by considering the basic type r,

Proposition 11 *Let $conv_{2-1}$, $conv_{3-2}$ and $conv_{1-3}$ be the PCFS terms defined in the previous chapter. For every element $d \in D_r$ the following equalities hold:*
$va^1_r(d) \simeq va^2_r(E[\![conv_{2-1}]\!]_\perp(d))$
$va^2_r(d) \simeq va^3_r(E[\![conv_{3-2}]\!]_\perp(d))$
$va^3_r(d) \simeq va^1_r(E[\![conv_{1-3}]\!]_\perp(d))$

Proof The proposition is an easy consequence of propositions 3 and 7

The computable functions $E[\![\mathrm{conv}_{2-1}]\!]_\perp$, $E[\![\mathrm{conv}_{3-2}]\!]_\perp$ and $E[\![\mathrm{conv}_{1-3}]\!]_\perp$ can be extended in a uniform way to higher order functions. These extensions follow the same lines of the extensions outlined in the previous chapter, where PCFS terms had been considered instead.

Proposition 12 *Given two partial representation functions for real numbers $va'_r : D_r \to R$ and $va''_r : D_r \to R$ if there are two computable (continuous) functions $f : D_r \to D_r$ and $f' : D_r \to D_r$ such that:*
$\forall d \in D_r. va''_r(d) \;\simeq\; va'_r(f(d)) \quad \textit{and} \quad va'_r(d) \;\simeq\; va''_r(f'(d))$
then for every type $\sigma \in T'$ there are two computable (continuous) $f_\sigma : D_\sigma \to D_\sigma$ and $f'_\sigma : D_\sigma \to D_\sigma$ such that:
$\forall d \in D_\sigma. va''_\sigma(d) \;\simeq\; va'_\sigma(f_\sigma(d)) \quad \textit{and} \quad va'_\sigma(d) \;\simeq\; va''_\sigma(f'_\sigma(d))$

Proof. The proof is similar to the proof of the analogous proposition 6 given in the previous chapter.

Corollary 13 *For every type σ the sets $\mathbb{R}^{d-i}_{\sigma}$ with $i \in \{1, 2, 3\}$ coincide, and contain the same sets of computable elements.*

In denoting the sets $I\!R_\sigma^{d-i}$, $i \in \{1,2,3\}$ it is therefore possible to drop the index i which refers to the real representation used. In the following we will write therefore simply $I\!R_\sigma^d$ to denote any one of the equal sets $I\!R_\sigma^{d-i}$.

From the point of view of computability theory the three representations are therefore equivalent. For the rest of the chapter we will limit ourselves to consider the negative digit representation.

4.4 Language completeness.

In this and the following sections we address the problem of connecting the two different notions of computability that we have introduced so far. As a first step in this direction we discuss whether the PCFS language is sufficiently expressive to denote all computable elements.

In following it is necessary to use recursive functions on natural numbers. Functions of several arguments on natural number can be handled as usual. The element $\overline{f}$ in D_σ, $\sigma = N \to N \to \ldots \to N$ represents the function $f : I\!N \times \ldots \times I\!N \to I\!N$ between natural numbers if

$$\overline{f}(n_1)\ldots(n_i) = n \text{ whenever } f(n_1 \ldots n_i) = n.$$

A closed term e in PCFS defines the function f on natural numbers if $E[\![e]\!]_\perp$ represents f. It is straightforward to show that for every partial recursive function f there is a closed PCFS term e defining f.

We consider first the case of real numbers.

Proposition 14 *A real number x is domain-computable if and only if there is a closed term e such that $x = va^3(E[\![e]\!]_\perp)$*

Proof. Let $\langle \cdot, \ldots, \cdot \rangle$ be any effective coding of the strings of natural numbers. Each natural number n can then be viewed as the code of $\langle n_0, \ldots, n_{n'} \rangle$.

Let nil be the natural number coding the empty string, nil $= \langle\ \rangle$; let car$'$, cdr$'$ and cons$'$ be the recursive functions such that:
car$'(n) = m$ iff $n = \langle n_0, \ldots, n_{n'} \rangle$ and $m = n_0$,
cdr$'(n) = m$ iff $n = \langle n_0, \ldots, n_{n'} \rangle$ and $m = \langle n_1, \ldots n_{n'} \rangle$,
cons$'(l, n) = m$ iff $n = \langle n_0, \ldots, n_{n'} \rangle$ and $m = \langle l, n_0, \ldots, n_{n'} \rangle$;
let $e_{\text{car}'}$, $e_{\text{cdr}'}$ and $e_{\text{cons}'}$ be three PCFS terms defining the above functions.

Consider the following enumeration of a subset D' of $D_{S(N)}$
$en(n) = [n_0.n_1.\ldots.n_{n'}]$. The set D' contains the finite strings having all elements different from $\perp_N$. Observe that for every element d in $D_{S(N)}$ if va$^3(d)$ is defined then $d = \bigsqcup\{d' \mid d' \in D' \wedge d' \sqsubseteq d\}$.

We introduce then a PCFS term
"PR" with type $N \to S(N) \to S(N)$ such that:
PR $:= Y_{N \to S(N) \to S(N)}(\lambda g.\lambda n.\lambda s.\text{cons}(e_{\text{car}'}n)(g(e_{\text{cdr}'}n)(\text{cdr}s)))$

It is not difficult to show that if $en(n)$ and s are consistent elements then $\mathrm{PR}\, n\, s = en(n) \sqcup s$.

Finally we have straightforwardly that if x is a computable real number and $d_x \in D_{S(N)}$ is an element representing it, then there is a recursive function f_d such that:
$d_x = \bigsqcup\{en(f_d(i)) \mid i \in N\}$.
Now if e_{f_d} is a term representing the function f_d, we have:
$d_x = E[\![Y_{N\to S(N)}(\lambda g.\lambda n.PR(e_{f_d}\, n)(g\mathrm{succ}\, n)))k_0]\!]_\perp$ and hence
$x = \mathrm{va}^3(E[\![e_{f_d}]\!]_\perp)$
□

We will extend the previous proposition to the set of the computable functions in $R_{r\to r}$. This will imply that our language is complete with respect to the set of computable functions definable via domain theory. This completeness is not so obvious. The language PCFS in fact is not complete with respect to the computable elements in $D_{r\to r}$. There are computable elements in $D_{r\to r}$ not definable by any PCFS term (see [21]).

Proposition 15 *An element $f : R^d_{r\to r}$ is computable if and only if there is PCFS term e such that $f = va^3(E[\![e]\!])$*

Proof. In this proof we use all the machinery introduced in the previous proof.

We introduce the following PCFS:
"cdar" with type $N \to S(N) \to N$
$\mathrm{cdar} := Y_{N\to S(N)\to N}\lambda g.\lambda n.\lambda s.(g(\mathrm{pred}\, n)(\mathrm{cdr}\, s))$, the intending meaning of
$(\mathrm{cdar}\ n\ s)$ is the n-th component of the string s;
"trunc" with type $N \to S(N) \to N$
$\mathrm{trunc} := Y_{N\to S(N)\to N}(\lambda g.\lambda n.\lambda s.e_{\mathrm{cons}'}(\mathrm{cdar}\, n\, s)(g(\mathrm{pred}\ n)(\mathrm{cdr}\, s)))$
the intending meaning being:
$\mathrm{trunc}\, n\, s = m$ if $m = \langle m_0, \ldots, m_n\rangle$ and $[m_0.\ldots.m_n] \sqsubseteq s$
$\mathrm{trunc}\, n\, s$ diverges if there is no such m,
that is, $\mathrm{trunc}\, n\, s$ returns the code of a finite string having n elements and approximating s, if it exists, otherwise $\mathrm{trunc}\, n\, s$ diverges.

The following fact can be show straightforwardly
If an element $d_f \in D_{S(N)\to S(N)}$ is computable then there is a recursive function f_d such that:
$en(f_d(n)) = \bigsqcup\{d' \mid d' \in D' \ \wedge\ d' \sqsubseteq d_f(en(n))\}$.

Now we have all the machinery necessary to prove the theorem. Given a computable function $f : \mathbb{R}^d_{r\to r}$ we construct a PCFS terms e_f such that $f = \mathrm{va}^3(E[\![e_f]\!])$.

Let d_f be a computable elements in $D_{r\to r}$ such that: $f = \mathrm{va}^3_{r\to r}(d_f)$ and let $f_d : \mathbb{N} \to \mathbb{N}$ be the recursive function such that:
$en(f_d(n)) = \bigsqcup\{d' \mid d' \in D' \ \wedge\ d' \sqsubseteq d_f(en(n))\}$
and let e_{f_d} be a PCFS term with type $N \to N$ defining the function f_d.

Consider now the following PCFS term e_f: $S(N) \to S(N)$
$e_f = Y_{N\to S(N)\to S(N)}(\lambda g.\lambda n.\lambda s.\mathrm{PR}(e_{f_d}(\mathrm{trunc}\, n\, s))(g(\mathrm{succ}\, n)s))k_0$.
It is not difficult to show that the term e_f satisfies the following equality:
$\forall d \in D'.E[\![e_f]\!]_\perp(d) = \bigsqcup\{d' \mid\ d \in D' \ \wedge\ d' \sqsubseteq d_f(d)\}$.

It remains to prove that $\mathrm{va}^3_{r\to r}(E[\![e_f]\!]) = f$. We prove the equivalent fact that for all $s \in D_r$ if $\mathrm{va}^3_r(s)$ is defined then $E[\![e_f]\!]_\perp(s) = d_f(s)$. In fact:
$E[\![e_f]\!]_\perp(s) = E[\![e_f]\!]_\perp(\bigsqcup\{d' \mid\ d' \in D' \ \wedge\ d' \sqsubseteq s\}) =$
$\bigsqcup\{E[\![e_f]\!]_\perp(d') \mid\ d' \in D' \ \wedge\ d' \sqsubseteq s\} =$
$\bigsqcup\{E[\![e_f]\!]_\perp(d') \mid\ d' \in D' \ \wedge\ d' \sqsubseteq s\} =$
$\bigsqcup\{d_f(d'') \mid\ d' \in D', d'' \in D' \ \wedge\ d' \sqsubseteq s \ \wedge\ d'' \sqsubseteq d_f(d')\} =$
$\bigsqcup\{d_f(d'') \mid\ d'' \in D' \ \wedge\ d'' \sqsubseteq d_f(s)\} =\ d_f(s)$
and this concludes the proof.
□

The previous result can be extended to functions on several arguments. Since we have not introduced the product type, functions having several arguments must be handled using currying. The set of functions $(\mathbb{R} \times \ldots \times \mathbb{R}) \to \mathbb{R})\ldots)$ is clearly isomorphic to the set of functions $\mathbb{R} \to (\mathbb{R} \to \ldots(\mathbb{R} \to \mathbb{R})\ldots)$. Via this isomorphism we can represent the function

Proposition 16 *A function $f : \mathbb{R}^d_{r\to(r\to\ldots(r\to r)\ldots)}$ is computable if and only if there is PCFS term e such that $f = va^3(E[\![e]\!]_\perp)$.*

The proof follows the same pattern of the previous proof.

It is an open problem to check if the proposition can be extended to higher order functions.

We make the following conjecture:

the functional that yields the maximum value of a function in the interval $[0, 1]$ is domain-computable but cannot be defined by any term in PCFS.

4.5 Some properties of computable functions on real numbers

In this section we want to define some properties of the computable functions. The properties will be mainly of topological flavour.

The function va^3_r is a partial function and its domain is the set $D^\dagger_r$ presented in definition 10.

In the following $D^\dagger_r$ will be considered a topological space endowed with the subset topology of Scott-topology. With some ambiguity the symbol va^3 will denote also the obvious total function from $D^\dagger_r$ to $\mathbb{R}$.

An important property of the function $\mathrm{va}^3 : D^\dagger_r \to \mathbb{R}$ is the following:

Proposition 17 *The function $va^3 : D^\dagger_r \to \mathbb{R}$ is continuous and surjective.*

Proof It is trivial to prove that the function va^3 is surjective.
To prove the continuity we prove that for every $s \in D_r^\dagger$ and for every open neighbourhood O of $\mathrm{va}^3(s)$ there is an open neighbourhood O' of s such that $\mathrm{va}^3(O') \subseteq O$.
In fact let $[s_0, s_1, \ldots]$ be an element in $D_r^\dagger$ and let O be an open set in $\mathbb{R}$ such that $\mathrm{va}^3([s_0, s_1, \ldots]) \in O$ then there is an ϵ such that
$\forall x \in \mathbb{R}.\ \mid x - \mathrm{va}^3([s_0, s_1, \ldots] \mid < \epsilon \ \Rightarrow\ x \in O$. Let n be a natural number such that $2^{-n} < \epsilon$. Then for every $s' \in D_r^\dagger.\ [s_0, \ldots s_n] \sqsubseteq s'$ we have $\mathrm{va}^3(s') \in O$ and from this the proof.
□

It is possible to give also a stronger result.

A topological definition is necessary.

Let $\equiv$ be an equivalence relation on a topological space S, the *quotient topology* on $S/\equiv$ is the finest topology which makes continuous the canonical map $c : S \to S/\equiv,\ \ c(x) = x/\equiv$.

Let $\equiv_{\mathrm{va}^3}$ indicate the equivalence relation on $D_r^\dagger$ induced by the function va^3.

Proposition 18 *$\mathbb{R}$ with the euclidean topology is equivalent to the quotient space $D_r^\dagger/\equiv_{va^3}$.*

Proof. To prove this it is sufficient to prove that the euclidean topology on $\mathbb{R}$ is the finer topology for which the function va^3 is continuous. This amounts to show that a set $O \subseteq \mathbb{R}$ is open in the euclidean topology if an only if it is the image, through va^3, of an open set O' in $D_r^\dagger$ such that:
$\forall d' \in O'.\forall d \in D_r^\dagger.\mathrm{va}^3(d) = \mathrm{va}^3(d') \Rightarrow d \in O'$. This can be proved as follows.

By proposition 17 if O is an open set in $\mathbb{R}$ then $\mathrm{va}^{-1}(O)$ is an open set of $D_r^\dagger$ and obviously
$\forall d' \in \mathrm{va}^{-1}(O).\forall d \in D_r^\dagger.\mathrm{va}^3(d) = \mathrm{va}^3(d') \Rightarrow d \in \mathrm{va}^{-1}(0)$.

Inversely given an open subset $O' \subseteq D_r^\dagger$ such that
$\forall d' \in O'.\forall d \in D_r^\dagger.\mathrm{va}^3(d) = \mathrm{va}^3(d') \Rightarrow d \in O'$,
and given $x \in \mathrm{va}^3(O')$, we construct the following sequence:
$s_0 := \mathrm{round}(x)$
$s_{i+1} := \mathrm{round}(x \times 2^{i+1}) - (\sum_{j=0}^{i} s_j \times 2^{i+1-j}) + 1$
a simple calculation can show that $\mathrm{va}^3([s_0, s_1, \ldots]) = x$, hence $[s_0, s_1, \ldots] \in O'$, then there is a n such that $\forall s' \in D_r^\dagger.\ [s_0, \ldots, s_n] \sqsubseteq s' \ \Rightarrow\ s' \in O'$,
it is the easy to see that the set
$\{x \mid x = \mathrm{va}^3(s'), [s_0, \ldots s_n] \sqsubseteq s'$ is contained in $\mathrm{va}^3(O')$ and is an open neighbourhood of x. From the generality of the choice of x it follows of that $\mathrm{va}^3(O')$ is an open set.
□

Topology is also useful to state some properties of the computable functions. A well know result of recursive analysis is that all computable functions are continuous.

Proposition 19 *All the functions in $\mathbb{R}^d_{r\to r}$ are continuous*

Proof Let f be a function in $\mathbb{R}^d_{r\to r}$ then there is an element $g \in D^\dagger_{r\to r}$ such that $\forall s \in D^\dagger_r.\ f \circ \mathrm{va}^3(s) = \mathrm{va}^3 \circ g(s)$.
Given an open set O of $\mathbb{R}$ we have $f^{-1}(O) = \mathrm{va}(g^{-1}(\mathrm{va}^{-1}(O)))$. It is easy to show that the set $g^{-1}(\mathrm{va}^{-1}(O))$ is open and satisfies condition
$\forall d' \in g^{-1}(\mathrm{va}^{-1}(O)).\ \forall d \in D^\dagger_r.\ \mathrm{va}^3(d) = \mathrm{va}^3(d') \Rightarrow d \in g^{-1}(\mathrm{va}^{-1}(O))$ from Proposition 18 follows that $\mathrm{va}(g^{-1}(\mathrm{va}^{-1}(O))) = f^{-1}(O)$ is open and therefore the thesis.
□

4.6 Relation between domain-computability and language-computability

So far two notions of computability on reals have been presented, one utilizing a programming language and one based on domain theory. As we will prove in the following, the two notions are strictly related.

Proposition 20 *A real number r is domain-computable if and only r is language-computable.*

The proof is an easy application of the proposition 14 of language completeness.

On the other hand the set of domain-computable functions from $\mathbb{R}$ to $\mathbb{R}$ is intrinsically different from the set of language computable functions. The functions contained in the two sets have different domains and codomains: the whole real line for $\mathbb{R}^d_{r\to r}$, while only the computable real numbers for $\mathbb{R}^l_{r\to r}$.

A strict relation between the two definitions can still be expressed as will be clear from the following.

Following the standard notation given a function $f : S \to T$ and a subset S' of S we denote with $f\,|_{S'}$ the restriction of f to the domain S'.

Proposition 21 *For every PCFS closed term e of type $r \to r$ the following equality holds:*
$R[\![e]\!] = va^3(E[\![e]\!])\,|_{\mathbb{R}^l}$

The proof is tedious but ultimately routine and is left to the reader.

This proposition shows how to go from the interpretation in $\mathbb{R}^d$ to the interpretation in $\mathbb{R}^l$. It is possible to go also in the opposite direction. To illustrate this we need to introduce some topological notions, first.

The set $\mathbb{R}^l$ of computable real numbers can be clearly viewed as a metric space, the metric being the euclidean one. An immediate consequence of propositions 19 is therefore:

Proposition 22 *All the functions in $\mathbb{R}^l_{r\to r}$ are continuous.*

Proof Follows immediately from Proposition 19.

The set $\mathbb{R}^l$ of the computable real number is clearly a dense subset of the real numbers (it contains all the rationals).

From general topology we have that if S, S' and T are three metric spaces and if S' is a dense subspace of S then every continuous function $f : S \to T$, is uniquely determined by its restriction to the set S', $f \mid_{S'} : S' \to T$, and moreover f can be obtained from $f \mid_{S'}$ by continuous extension. Given a function $g : S' \to T$ we denote with $Ex_S(g)$ its continuous extension to the domain S, if it exists. Formally $f = Ex_S(f \mid_{S'})$.

Proposition 23 *For every PCFS closed term e of type $S(N) \to S(N)$ the following equality holds:*
$va^3_{r\to r}(E[\![e]\!]_\perp) = Ex(R_{r\to r}[\![e]\!])$

Proof Follows immediately from Proposition 21.

From Proposition 15 we have also that the functionals of "domains restriction" and "continuous extension" define a bijection between the set of language computable real functions and the domain-computable real functions

The above results can be easily generalized to functions in several arguments. Also in this case functions in several arguments are treated by currying.

A function $f : \mathbb{R} \to (\mathbb{R} \to \ldots (\mathbb{R} \to \mathbb{R}) \ldots)$ is defined to be continuous if the isomorphic function $\overline{f} : (\mathbb{R} \times \ldots \times \mathbb{R}) \to \mathbb{R}) \ldots)$ is continuous. (Remark. This amounts to say that f is continuous when the function spaces are endowed with the compact open topology.)

Given a function $f : \mathbb{R} \to (\mathbb{R} \to \ldots (\mathbb{R} \to \mathbb{R}) \ldots)$, we denote with $f \mid \mathbb{R}^l$ the function in $\mathbb{R}^l \to (\mathbb{R}^l \to \ldots (\mathbb{R}^l \to \mathbb{R}) \ldots)$ defined in the obvious way.

Proposition 24 *i) For every PCFS closed term e of type $r \to (r \to \ldots (r \to r) \ldots)$ the following equality holds:*
$R[\![e]\!] = va^3(E[\![e]\!]_\perp) \mid_{\mathbb{R}^l}$

ii) All the functions in $\mathbb{R}^d_{r\to(r\to\ldots(r\to r)\ldots)}$ are continuous.

iii) All the functions in $\mathbb{R}^l_{r\to(r\to\ldots(r\to r)\ldots)}$ are continuous.

iv) For every PCFS closed term e of type $r \to (r \to \ldots (r \to r) \ldots)$ the following equality holds:
$va^3(E[\![e]\!]_\perp) = Ex(R[\![e]\!])$

v) A function $f : \mathbb{R}^d_{r\to(r\to\ldots(r\to r)\ldots)}$ is computable if and only if there is a function $f' : R^l_{r\to(r\to\ldots(r\to r)\ldots)}$ such that $f = Ex(f')$

The proofs of the above propositions are not difficult. In some cases they are an easy generalization of the corresponding proofs for the unary function case.

Chapter 5

A domain of approximations for real numbers.

5.1 Introduction.

The domains that we used in the previous chapter to study computability on real numbers were constructed so as to give a denotational semantics to PCFS.

In the literature there are other approaches to computability on real numbers which make use of different sorts of domains.

In one of his early papers on domain theory, Scott [26] suggests that a cpo composed of intervals of the real line can be used to study computability on real numbers. Previously also Martin-Löf [16] constructed a similar space of approximations. A similar idea was also presented in Lacombe [15]. The real line can be embedded in these spaces of approximations where a notion of computability can be defined in a natural way. Many results concerning the computability theory on real numbers are given in these contexts.

These spaces of approximations are particular cases of "countably based complete partial orders" ccp's whose formal theory has been developed in Smyth [28].

Weihrauch and Schreiber [34] developed similar ideas in the context of algebraic cpo's enriched with a notion of distance and weight.

In the following we present a construction that is similar in many aspects to the ones mentioned above but which has also some important differences.

In constructing a space of approximations a given form of real number representation is always assumed. All space constructions mentioned above are based on the representation of real numbers as converging sequences of rational intervals (definition 1). This form of representation is not appropriate to be

used for implementations of real number computation. One can see this informally, by considering that the efficiency of the computation is certainly decreased by the existence of too many approximation points (every rational interval is an approximation point), i.e. cumbersome representations.

In view of our goals we base our construction on other forms of real number representations: the digit sequence and the integer sequence representations of definition 2 c) and d). These forms are in fact more suitable to be used in an actual implementation.

A second important difference is the following: our space of approximations turns out to be a Scott-Domain. The other approaches use instead more general forms of cpo's, which are less used in denotational semantics.

5.2 The construction of the domain RD

The domain of approximation defined next is called Reals Domain (RD).

The importance of RD is twofold. First the proofs of properties of the computable functions on real number are simpler when RD is used. Moreover in the chapter 7 RD will be a source of inspiration for the definition of a new functional language for real number computation.

We first present a construction of RD starting from the binary negative digit notation of real numbers. Later we will show that RD can be obtained also repeating the same construction starting from the Cauchy sequence notation of real numbers.

Let s be a sequence of integers defining a real number r according to definition 2 (the binary negative digit notation) . And let t be an initial subsequence of s. t gives partial information about the value r. Examining t we can derive that the value r is contained in an interval of real numbers. For example the sequence $2 : -1$ is the initial notation of a number contained in the closed interval $[1, 2]$. All the sequences beginning with $2 : -1$ denote a real contained in $[1, 2]$ and each number in $[1, 2]$ can be denoted by a sequence beginning with $2 : -1$.

This leads to the definition of a functions from the finite sequence of integers to intervals in the real line. To any finite sequence $t_0 : t_1 : \ldots : t_n$ we associate the interval $[b, c]$ containing the real numbers that can be represented by sequences having as initial subsequences $t_0 : t_1 : \ldots : t_n$.

More formally we define a function $\text{quot}_\top$ from the set of finite elements of $D_{S(N)}$, $D^\circ_{S(N)}$, to the set of the closed intervals of real line. In the following RI will denote the set of closed intervals of real line.

Before giving the definition of $\text{quot}_\top$ it is useful to define the extension of the arithmetic operation to RI and the extension of the order relation to RI.

Definition 12 *The arithmetic operation on RI are defined by:*
$[a, b] + [a', b'] := [a + a', b + b']$
$-[a, b] := [-b, -a]$
$[a, b] \times [a', b'] := [\min\{a \times a', a \times b', b \times a', b \times b'\}, \max\{a \times a', a \times b', b \times a', b \times b'\}]$

$$1 \div [a,b] := \begin{cases} [-\infty,+\infty] & \textit{if } 0 \in [a,b] \\ [1 \div b, 1 \div a] & \textit{otherwise} \end{cases}$$

A partial order relation $\leq$ is defined on RI by
$[a,b] \leq [a',b']$ *if* $b \leq a'$.

An informal justification for the above definition is the following.
$[a,b] + [a',b']$ is the interval of values that are obtained adding an element of $[a,b]$ to an element of $[a',b']$, and similarly for the other arithmetic operations.

Notation If r is a real number and $[a,b]$ is a closed interval in $\mathbb{IR}$, we use the abbreviation $r + [a,b]$ to denote the interval $[r,r] + [a,b]$. Similar abbreviation will be used also for the other arithmetic operation and for the order relation.

Definition 13 *The function $quot_\top : D_r \to RI$ is recursively defined by:*

$$quot_\top([\,]) := [-\infty,+\infty]$$

$$quot_\top([t_0.t_1 \ldots t_i]) := \begin{cases} [-\infty,+\infty] & \textit{if } t_0 = \bot \\ z(t_0) + (quot'_\top[t_1 \ldots t_i]) & \textit{otherwise} \end{cases}$$

$$quot'_\top([\,]) := [-1,1]$$

$$quot'_\top([t_0.t_1 \ldots t_i]) := \begin{cases} [-1,1] & \textit{if } t_0 = \bot \\ (t_0 - 1 + quot'_\top[t_1 \ldots t_i]) \div 2 & \textit{if } t_0 \in \{0,1,2\} \\ \textit{the empty interval} & \textit{otherwise} \end{cases}$$

where $z : \mathbb{N} \to Z$ is the effective enumeration of the integers defined in section 3.2.

If $t = [t_0.t_1 \ldots t_i]$ and $\forall h\, 0 \leq h \leq i.\, t_i \neq \bot$ then the left and the right limit of the interval $\mathrm{quot}_\top(t)$ denote respectively the smallest and the largest number that can be denoted by an element greater than t (in the domain order). And every number inside the interval $\mathrm{quot}_\top(t)$ can be denoted by a proper element greater than t.

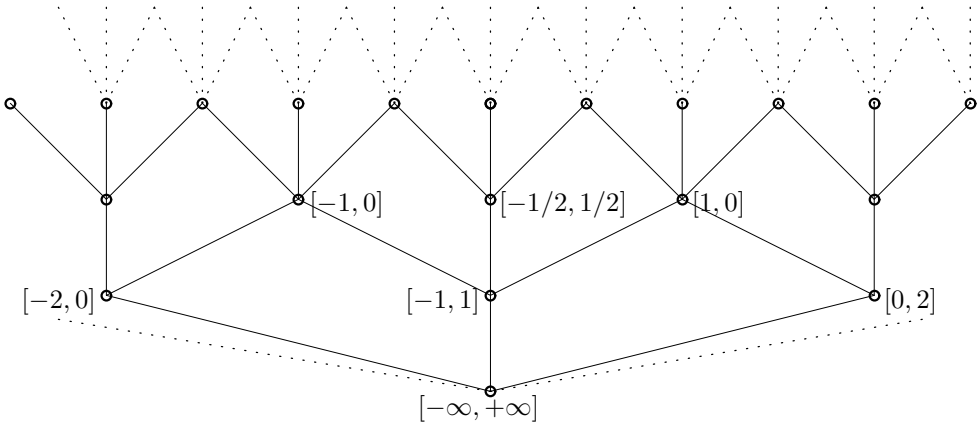
Examples:
$\mathrm{quot}_\top([0]) = [-1,1]$, $\mathrm{quot}_\top([0.1]) = [0,1]$, $\mathrm{quot}_\top([0.1.0]) = [1/4, 3/4]$.

It is not difficult to verify that the image of the function $\mathrm{quot}_\top$ is constituted by the interval $[\infty,+\infty]$, by the empty interval and by the intervals having form $[(j-1)/2^i, (j+1)/2^i]$ where i is a natural number and j is an integer.

The rational numbers in the form $i/2^j$ with $i \in \mathbb{N}$, $j \in Z$ are called dyadic rationals. We call dyadic intervals the rational intervals having the form $[-\infty,+\infty]$ or $[(j-1)/2^i, (j+1)/2^i]$ where i is a natural number and j is an integer.

Let $DI_\top$ denote the partial order containing the dyadic intervals and the empty interval. The order relation on $DI_\top$ is the superset relation.

The diagram representing $DI_\top$ has form:



To extend the function $\text{quot}_\top$ to all the elements of $D_{S(N)}$ it is necessary to complete the partial order $DI_\top$.

Definition 14 *Let $(RD_\top)$ denote the cpo obtained by the ideal completion of $DI_\top$.*

Proposition 25 *$RD_\top$ is a consistently complete ω-algebraic cpo.*

Proof $RD_\top$ is obviously an ω-algebraic cpo.

It is easy to observe the intersection of any pair of intervals in $RI_\top$ is again an interval in $RI_\top$. From this the fact that $RD_\top$ is consistently complete follows easily.

□

The function $\text{quot}_\top$ is monotone and therefore it can be extended by continuity to a a function $\overline{\text{quot}}_\top : D_{S(N)} \to RD$;

The domain $RD_\top$ can be thought as composed by equivalence classes of elements $D_{S(N)}$. The equivalence classes of finite elements are composed by elements containing identical information about the real value they approximate (via the binary negative digit notation).

It is interesting to observe that $RD_\top$ can be obtained also repeating the previous construction using a different notation for the real numbers. Instead of the binary negative digit notation, the Cauchy sequence notation of reals presented in definition 5 R^2 can be considered.

The new construction leads to the definition of exactly the same domain $RD_\top$ and of a function $\overline{\text{quot}}_1 : D_{S(N)} \to RD_\top$ which maps each element in $D_{S(N)}$ to the interval of real numbers that it approximates via the Cauchy sequence notation. $\overline{\text{quot}}_1$ is the continuous extension of the functions quot_1 defined by:

$\mathrm{quot}_1([\,]) = \bot$

$\mathrm{quot}_1([t_0.t_1.\ldots.t_i]) = [j,k] \sqcup \mathrm{quot}_1([t_1.\ldots.t_i]) \div 2$ where $j = (z(t_0) - 1)$ and $k = (z(t_0) - 1)$.

The empty interval, that is the maximum element in $RD_\top$ does not approximate any real number. It has been introduced to denote "inconsistent" sequences, that is, sequences that not approximate or denote any real number.

In the following we prefer to employ a domain having just consistent elements. This can be done removing the empty interval from $RD_\top$.

Formally:

Definition 15 *Let DI denote the partial order of the dyadic intervals with the superset relation.*

Let (RD) denote the cpo obtained by the ideal completion of $DI_\top$.

It is easy to check that $RD_\top = RD \cup \{$ empty interval $\}$ and that RD is a consistently complete ω-algebraic cpo.

The function $\overline{\mathrm{quot}}_\top : D_{S(N)} \to RD_\top$ induces in a natural way a partial function $\overline{\mathrm{quot}} : D_{S(N)} \to RD$.

5.3 Infinite elements.

We intend to investigate in this section the relation existing between the set of infinite elements of RD and the real line.

It is not difficult to prove that:

Proposition 26 *For every $\alpha \in D_{S(N)}$, $R^3[\![\alpha]\!]$ is defined if and only if $\overline{quot}_\top(\alpha)$ is an infinite element.*

For every α, β in $D_{S(N)}$

$$\overline{quot}_\top(\alpha) = \overline{quot}_\top(\beta) \;\Rightarrow\; va^3(\alpha) \simeq va^3(\beta)$$

Proof. Straightforward.

The implication

$$\mathrm{va}^3(\alpha) \;=\; \mathrm{va}^3(\beta)) \;\Rightarrow\; \overline{\mathrm{quot}}_\top(\alpha) = \overline{\mathrm{quot}}_\top(\beta)$$

is not always true. For every dyadic rational number j we can divide the sequences denoting it in three non-empty classes: the class of the sequences eventually ending with a series of 1, the class of those ending with a series of 0 and the class of those ending with a series of 2. The elements contained in each class are identified by the functions $\overline{\mathrm{quot}}_\top$. We will call $\overline{j}$, $\overline{j^-}$ and $\overline{j^+}$ respectively these representations in RD. $\overline{j}$, $\overline{j^-}$ and $\overline{j^+}$ are distinct elements of RD. In the order of RD we have:

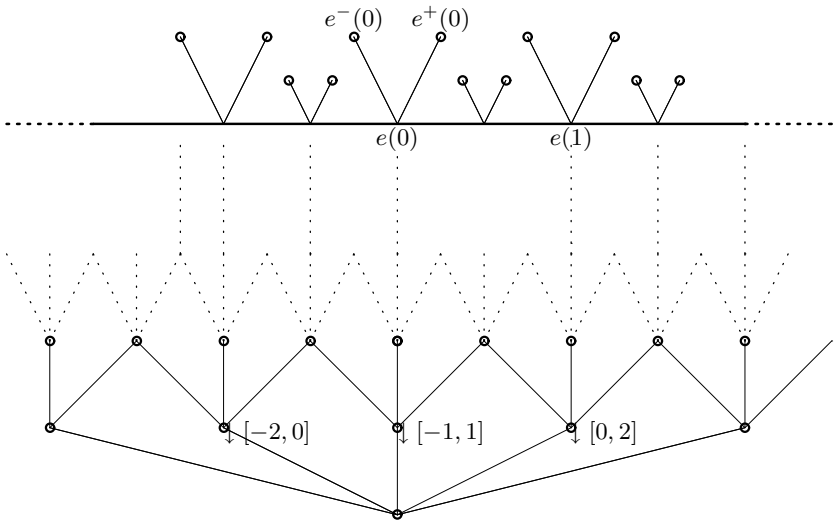
$\overline{d} \sqsubseteq \overline{d^-}$, $\overline{d} \sqsubseteq \overline{d^+}$

and $\overline{d^-}$ incomparable with $\overline{d^+}$

For non rational dyadic numbers we have the following property:
for every $\alpha, \beta \in D_{S(N)}$

$$(\mathrm{va}^3(\alpha) = \mathrm{va}^3(\beta) = r \ \wedge \ r \text{ not rational dyadic }) \ \Rightarrow \ \overline{\mathrm{quot}}_\top(\alpha) = \overline{\mathrm{quot}}_\top(\beta).$$

Moreover any infinite point in RD is the representation of a real number.
The set of infinite elements in RD looks like the real line but for the dyadic numbers. Each is triplicated, in fact.
RD can be represented by the following diagram.



In the next section we will show how to solve the problem of multiple representations by means of a retract construction.

RD is an effective Scott-domain.

Proposition 27 *Let $\langle\ \rangle$ be any coding function of triples of natural numbers. We can define an effective enumeration of the finite elements of RD in the following way:*

$$en_r(0) = \perp$$

$$en_r(\langle m, m', n\rangle + 1) = [(m - m' - 1)/2^n, (m - m' + 1)/2^n].$$

$(RD, \sqsubseteq, en_r)$ is then an effective Scott domain.

Proof. Straightforward.

Since RD is an effective Scott-Domain we can apply to it the standard machinery for defining computability. We can obtain in this way another definition of computability for real numbers.

Chapter 6

Topological characterizations.

In this chapter we present some results concerning the topological relationship between the real line and the Scott-domain RD. These results are then generalized to the functions spaces.

Using RD we give also a new definition of computable real function. We present then equivalence results which link the new notion of computability with the old one.

Topologically the domain RD is much more tightly related with the real line that the domain $D_{S(N)}$. And this fact is true also for function spaces. As a consequence, the use of RD makes it easier to prove the topological properties of the computable function on the reals.

The main topological relation considered in this section is the retraction between spaces. The real line turns out to be a retract of the subspace of infinite elements of RD.

6.1 Topological preliminaries

For completeness we give here important topological notions that will be use in the following.

Let S be a subset of a topological space T, the *subspace* topology on S is defined by: a set O in S is an open set in the subspace topology if and only if there is an open set O' in T such that $O = S \cap O'$

Let Q be an equivalence relation on a space T. Let ϕ denote the canonical map from T to T/Q, $\phi(x) = [x]$. The *quotient topology* on T/Q is defined by: a set O in T/Q is open if and only if $\phi^{-1}(O)$ is open.

A space S is said to be a *retract* of a space T if there are two continuous functions $q : T \to S$ and $e : S \to T$ such that $q \circ e = Id_S$.

In this case the following propositions are true:
1) S is isomorphic to the subspace $e(S)$ of T
2) Let Q denote the equivalence relation induced on T by q, S is isomorphic to the quotient space T/Q.

A *prebase* P of a space T is a family of open sets of T such that any other open set O of T can be written as union of finite intersections of sets in P.

Let X be a set, given a family P of subsets of X there is a unique topology on X such that P forms a prebase for that topology.

Let S and T be two topological spaces, and $S \to T$ be set of the continuous functions from S to T. The *compact-open* topology on $S \to T$, is the topology having as prebase the sets of the form $\{f \mid f(c) \subseteq o\}$, where c is a compact set of S and o is an open set of T.

6.2 The topological relations between the domain RD and the real line.

Let $RD^\dagger$ denote the subspace of RD consisting of the infinite elements.

Proposition 28 *The real line is a retract of $RD^\dagger$ via a pair of continuous functions $q_r : RD^\dagger \to I\!R$ and $e_r : I\!R \to RD^\dagger$.*
$q_r(\overline{x}) := x$ *if x is not a dyadic number*
$q_r(\overline{x}) := q_r(\overline{x^+}) := q_r(\overline{x^-}) := x$ *if x is a dyadic number*
$e_r(x) := \overline{x}$

The proof is easy and is left to the reader.

It is also easy to prove that for every s in $D_{S(N)}$ the following equality holds $\mathrm{va}^3(s) \simeq q_r(\overline{\mathrm{quot}}(s))$.

The function q_r associates to each element of $RD^\dagger$ the corresponding real number. We can interpret e_r as the function which picks a canonical representative for each real number.

Using q_r it is possible to give a new definition of computable real number: a real number x is RD-computable if there is a computable element $d \in RD$ such that $x = q_r(d)$. It is not difficult to prove that the new definition coincides with definitions of domain-computable and language-computable real number given in the previous chapters.

Using e_r and q_r it is possible to associate to each Scott-continuous function $f : RD \to RD$ a partial real function $\overline{f} : I\!R \to I\!R$ defined by $\overline{f} := q_r \circ f \circ e_r$, $\overline{f}$ is partial because q_r is not always defined. We define $\overline{f}$ as the function on reals represented by f.

We obtain in this way a new definition of computable function on real numbers.

Definition 16 *A (partial) function $g : \mathbb{IR} \to \mathbb{IR}$ is RD-computable if there exits a computable function $f : RD \to RD$ such that $g = \overline{f}$.*

Thus we have so far we presented, using domain theory, two different definitions of computable function on real numbers. That is: the definition of domain-computable function and the one above.

The definition of RD-computable function allows to associate to every element f in $RD \to RD$ a function on real numbers. If the element f is not sufficiently defined the associated function is a partial function. This is not true for the definition of domain-computable function. In this case in fact only a subset of the elements of $D_{S(N)\to S(N)}$ induce a function on reals via the interpretation functions va^i and the funtions in the images of va^i are total functions. In general elements in $D_{S(N)\to S(N)}$ do not preserve the extensional equality of representations.

The two definitions however give the same set of total computable functions.

Proposition 29 *A total function $g : \mathbb{IR} \to \mathbb{IR}$ is domain-computable if and only if it is RD-computable.*

Proof. Omitted.

For every Scott-continuous function $f : RD \to RD$, the function $\overline{f}$ is a composition of continuous functions and therefore is continuous. In this way we obtain a new proof of a classical result in computable analysis: every computable functions on real numbers is continuous w.r.t. the Euclidean topology.

In the following we extend the notion of RD-computability to functions in several arguments and to higher order functions. The method followed in this extension is substantially different from the one employed in defining the class of domain-computable functions.

We will show how the retract relation existing between $\mathbb{IR}$ and $RD^\dagger$ can also be extended to function spaces.

However the retract relation cannot be defined for arbitrary higher order functions spaces, see follow.

As in previous chapters we consider the set T' of types on reals. T' is generated by the grammar: $\sigma := r \mid \sigma \to \sigma$

In the following we will use the function *rank* $(T' \to \mathbb{N})$. The rank of a type σ, $\partial(\sigma)$, is defined by:

$$\partial(r) := 0$$

$$\partial(\sigma \to \sigma') := max\{\partial(\sigma) + 1, \partial(\sigma')\}$$

The rank of a type measures how "higher order" the type really is.

Note that all types σ with $\partial(\sigma) = 1$ have the form $\sigma = r \to (r \to \ldots r) \ldots)$.

Notation. In the following we denote by T'_n the set of type $\sigma \in T'$ such that $\partial(\sigma) = n$.

We can define the retract relation just for function spaces having rank 1. For function spaces of rank 2, a set theoretical relation is stated. We do not introduced here definitions for functionals on reals having rank larger than 2. This is not a severe limitation, in fact in analysis functionals having rank larger that 2 are almost never employed [24].

For each type $\sigma \in T'_1$, a topological space $\mathbb{R}_\sigma$ is defined by structural induction on σ.

$\mathbb{R}_r = \mathbb{R}$
$\mathbb{R}_{r\to\sigma} = \{f : \mathbb{R} \to \mathbb{R}_\sigma \mid \text{f total continuous }\}$
The topology on $\mathbb{R}_{r\to\sigma}$ is the compact open topology.

Remark. In analysis the compact open topology is the topology normally associated to the space of the real continuous functions.

We do not associate a topological space to the types of rank 2. The sets associated to these types are unstructured.

For each type $\sigma \in T'_2$ R_σ is defined by:
$\mathbb{R}_{\sigma_1\to(\ldots(\sigma_n\to r)\ldots)} = \{F : \mathbb{R}_{\sigma_1} \to (\ldots(\mathbb{R}_{\sigma_n} \to R)\ldots) \mid \lambda\langle x_1,\ldots,x_n\rangle.F(x_1)\ldots(x_n) : (\mathbb{R}_{\sigma_1} \times \ldots \times \mathbb{R}_{\sigma_n}) \to \mathbb{R}$ is a continuous function $\}$.

There are no sets of real functionals associated to types of rank larger than two.

A collection of effective Scott-domain RD_σ, one for each type $\sigma \in T'$ can then be defined by induction on σ
$RD_r := RD$
$RD_{\sigma\to\sigma'}$ is the domain of the Scott-continuous functions from RD_σ to $RD_{\sigma'}$.

In RD not every element denotes a real number, some elements in RD are just finite approximations of real numbers. Similarly not every element in RD_σ will represent an element in $\mathbb{R}_\sigma$. Hence we define for each type σ, having a rank strictly smaller than 3, a subspace $RD^\dagger_\sigma$ of the domain RD_σ. The elements in $RD^\dagger_\sigma$ will denote the elements in $\mathbb{R}_\sigma$.

$RD^\dagger_r := \{s \in RD \mid x \text{ is an infinite element of } RD\}$
$RD^\dagger_{\sigma\to\sigma'} := \{g \in RD_{\sigma\to\sigma'} \mid g(RD^\dagger_\sigma) \subseteq RD^\dagger_{\sigma'}\}$
The topology on $RD^\dagger_\sigma$ is the subspace topology of Scott-topology.

Notation. Given $s_\sigma \in RD_\sigma$ we denote with $\overset{\vee}{s}_\sigma$ the set $\{t_\sigma | t_\sigma \in RD_\sigma, s_\sigma \leq t_\sigma\}$ and with $\overset{\overline{\vee}}{s}_\sigma$ the set $\overset{\vee}{s}_\sigma \cap RD^\dagger_\sigma$

The retract relation can be extended to the rank 1 functions:

Proposition 30 *For each type $\sigma \in T_1'$ ($\sigma = r \to \sigma_1$), $\mathbb{R}_\sigma$ is a retract of $RD^\dagger_\sigma$. The pair of retract functions $q_\sigma : RD^\dagger_\sigma \to \mathbb{R}_\sigma$ and $e_\sigma : \mathbb{R}_\sigma \to RD^\dagger_\sigma$ is defined as follows:*

$q_\sigma(g) := q_{\sigma_1} \circ g \circ e_r$

$e_\sigma(f)$ is the continuous extension of the function $f' : RD^\circ \to RD_\sigma$ defined by:

$$f'(s^\circ) = \begin{cases} \bot & \text{if } s^\circ = \bot \\ \bigsqcup\{t^\circ \mid e_{\sigma_1} \circ f \circ q_r(\overline{s^\circ}^{\vee}) \subseteq \overset{\vee}{t^\circ}\} & \text{otherwise} \end{cases}$$

Proof.

We prove by structural induction on the type $\sigma \in T_1'$ the following properties:

i) if $\sigma = r \to \sigma_1$ then $\forall f \in \mathbb{R}_{r\to\sigma_1}.\ \forall\ s \in RD^\dagger.\ e_{r\to\sigma_1}(f)(s) = (e_{\sigma_1} \circ f \circ q_r)(s)$
ii) q_σ is a well defined function,
iii) q_σ is a continuous function,
iv) e_σ is a well defined function,
v) e_σ is a continuous function,
vi) $q_\sigma \circ e_\sigma = id_{\mathbb{R}_\sigma}$.

Note. In the following $[a, b]$ can indicate either a generic dyadic interval either a finite element in RD.

The basic step follows immediately from Proposition 28.

Inductive step. In this case σ has the form $\sigma = r \to \sigma_1$.

i) It is easy to check that for every $s \in RD^\dagger$ the set
$\{[a, b] \mid q_r(s) \in (a, b),\ [a, b]$ dyadic interval $\}$ is a system of neighborhoods for $q_r(s)$. By induction hypothesis $e_{\sigma_1} \circ f$ is a continuous function. So
$\{(e_{\sigma_1} \circ f)([a, b]) \mid q_r(s) \in (a, b),\ [a, b]$ dyadic interval $\}$
is a system of neighborhoods for $(e_{\sigma_1} \circ f \circ q_r)(s)$.
The following equality therefore holds:

$$\{t^\circ \mid (e_{\sigma_1} \circ f)([a, b]) \subseteq \overset{\vee}{t^\circ},\ q_r(s) \in (a, b),\ [a, b] \text{ dyadic interval } \} =$$
$$= \{t^\circ \mid (e_{\sigma_1} \circ f \circ q_r)(s) \sqsubseteq \overset{\vee}{t^\circ}\}.$$

We have therefore the following chain of equalities:

$$e_{r\to\sigma_1}(f)(s) = \bigsqcup\{t^\circ \mid \underline{(e_{\sigma_1}}_{\vee} \circ f \circ q_r)(\overline{s^\circ}^{\vee}) \subseteq \overset{\vee}{t^\circ},\ s^\circ \in RD^\circ,\ s^\circ \sqsubseteq s\} =$$
$$= \bigsqcup\{t^\circ \mid (e_{\sigma_1} \circ f \circ q_r)([a, b]) \subseteq \overset{\vee}{t^\circ},\ [a, b] \text{ dyadic inteval },\ [a, b] \sqsubseteq s\} =$$
$$= \bigsqcup\{t^\circ \mid (e_{\sigma_1} \circ f)([a, b]) \subseteq \overset{\vee}{t^\circ},\ [a, b] \text{ dyadic interval } q_r(s) \in (a, b)\} =$$
$$= \bigsqcup\{t^\circ \mid (e_{\sigma_1} \circ f \circ q_r)(s) \sqsubseteq \overset{\vee}{t^\circ}\}$$
$$= (e_{\sigma_1} \circ f \circ q_r)(s)$$

ii) In order to prove that $q_{r\to\sigma_1} : RD^\dagger_{r\to\sigma_1} \to RD_{r\to\sigma_1}$ is a well defined function it is sufficient to prove that for every $g : RD^\dagger_{r\to\sigma_1}$, $q_{r\to\sigma_1}(g) = q_{\sigma_1} \circ g \circ e_r$ is a total continuous function. This follows immediately from induction hypothesis.

iii) Since $q_{r\to\sigma_1}$ is continuous if and only if for $g \in RD^\dagger_{r\to\sigma_1}$ and for every neighborhood U of $q_{r\to\sigma_1}(g)$ it is possible to find an open neighborhood V of g such that $q_{r\to\sigma_1}(V) \subseteq U$. We will prove this second fact.

By definition of compact open topology, for any neighborhood U of $q_{r\to\sigma_1}(g)$ there is a finite set $\{c_1 \dots c_n\}$ of compact sets in $\mathbb{R}$ and a finite set $\{o_1 \dots o_n\}$ of open sets in $\mathbb{R}_{\sigma_1}$ such that:
1) $U \supseteq \bigcap_{1\le i\le n}(c_i \to o_i)$
2) $q_{r\to\sigma_1}(g) = q_{\sigma_1} \circ g \circ e_r \in \bigcap_{1\le i\le n}(c_i \to o_i)$
where with $c_i \to o_i$ we denote the set of functions $\{f : \mathbb{R}_{r\to\sigma_1} \mid f(c_i) \subseteq o_i\}$. By induction hypothesis q_{σ_1} is continuous and so is $q_{\sigma_1} \circ g$. Hence for every i $1 \le i \le n$ the set $(q_{\sigma_1} \circ g)^{-1}(o_i)$ is an open set of RD and hence using the definition of Scott-topology we have:
$(q_{\sigma_1} \circ g)^{-1}(o_i) = \bigsqcup\{\overset{\vee}{s^\circ_h} \mid s^\circ_h \text{ finite}, s^\circ_h \in (q_{\sigma_1} \circ g)^{-1}(o_i)\}$.
From 2) we have also $(q_{\sigma_1} \circ g \circ e_r)(c_i) \subseteq o_i$. That is $e_r(c_i) \subseteq (q_{\sigma_1} \circ g)^{-1}(o_i)$. So the set $\{\overset{\vee}{s^\circ_h} \mid s^\circ_h \text{ finite } s^\circ_h \in (q_{\sigma_1} \circ g)^{-1}(o_i)\}$ is a covering of $e_r(c_i)$. $e_r(c_i)$ is the image of a compact set and so is a compact set itself, this implies that there is a finite subcovering $\{\overset{\vee}{s^\circ_{i,1}}, \dots, \overset{\vee}{s^\circ_{i,m}}\}$ of the covering $\{\overset{\vee}{s^\circ_h} \mid s^\circ_h \text{ finite } s^\circ_h \in (q_{\sigma_1} \circ g)^{-1}(o_i)\}$.

Moreover $s_{i,j} \in (q_{\sigma_1} \circ g)^{-1}(o_i)$, that is $g(s_{i,j}) \in q^{-1}_{\sigma_1}(o_i)$ and since $q^{-1}_{\sigma_1}(o_i)$ is an open set in RD_{σ_1} we have:
$\forall s^\circ_{i,j}.\ \exists t^\circ_{i,j} \in RD^\circ_{\sigma_1}.\ t^\circ_{i,j} \sqsubseteq g(s^\circ_{i,j}) \wedge t^\circ_{i,j} \in q^{-1}_{\sigma_1}(o_i)$.

It is now possible to define the open neighborhood V of g we are looking for:
$V = \bigcap_{i,j}(s^\circ_{i,j} \overset{\vee}{\Rightarrow} t^\circ_{i,j})$
where $s^\circ_{i,j} \Rightarrow t^\circ_{i,j}$ denotes the obvious step function, and $(s^\circ_{i,j} \overset{\vee}{\Rightarrow} t^\circ_{i,j})$ the cone generated by it.

In fact it is easy to see that V is a neighborhood of g. To show that $q_{r\to\sigma_1}(V) \subseteq U$ we show that for every $g \in V$ we have $q_{r\to\sigma_1}(g) \in \cap_i(c_i \to o_i)$. In fact
$\forall i.\ \forall s \in c_i.\ \exists s^\circ_{i,j}.\ e_r(s) \in s^\circ_{i,j}$ and this implies that
$\forall i.\ \forall s \in c_i.\ q_{r\to\sigma_1}(g)(s) = q_{\sigma_1}(g(e_r(s))) \subseteq q_{\sigma_1}(g(\overset{\vee}{s^\circ_{i,j}})) \subseteq q_{\sigma_1}(\overset{\vee}{t^\circ_{i,j}}) \subseteq o_i$ that is
$\forall i.\ q_{r\to\sigma_1}(g) \in (c_i \to o_i)$.

iv) In order to proof that $e_{r\to\sigma_1}$ is well defined we need to prove that
$\forall f \in \mathbb{R}_{r\to\sigma_1}.\ e_{r\to\sigma_1}(f) \in RD^\dagger_{r\to\sigma_1}$. That is $e_{r\to\sigma_1}(f)$ is a well defined continuous function such that $e_{r\to\sigma_1}(RD^\dagger) \subseteq RD^\dagger_{\sigma_1}$.

$e_{r\to\sigma_1}(f)$ is a well defined continuous function $RD \to RD_{\sigma_1}$ because it is the continuous extension of a monotonic function $RD^\circ \to RD_{\sigma_1}$.

The prove that $e_{r\to\sigma_1}(f)(RD^\dagger) \subseteq RD^\dagger_{\sigma_1}$ follows easily from point i).

v) In order to prove the continuity of $e_{r\to\sigma_1}$ we use the two following properties:
1. a function $h : S \to T$ between topological spaces is continuous if given a prebase P for T we have:
$\forall s \in S.\ \forall p \in P.\ (\ h(s) \in p \Rightarrow \exists o$ open neighborhood of $s\ .\ h(o) \subseteq p)$

2. in the domain of Scott-continuous functions $D \to D'$ the upward cones of the finite step functions form a prebase.

The continuity of $e_{r\to\sigma_1}$ follows from the following chain of implications:
$e_{r\to\sigma_1}(f) \in s^\circ \overset{\vee}{\Rightarrow} t^\circ \iff t^\circ \sqsubseteq e_{r\to\sigma_1}(f)(s^\circ)$
$\iff (e_{\sigma_1} \circ \overline{f} \circ q_r)(\overset{\vee}{s^\circ}) \subseteq \overset{\vee}{t^\circ} \iff$
$(e_{\sigma_1} \circ \overline{f})(q_r(\overset{\vee}{s^\circ})) \subseteq \overset{\vee}{t^\circ} \iff$
$\overline{f}(q_r(\overset{\vee}{s^\circ})) \subseteq e^{-1}_{\sigma_1}(\overset{\vee}{t^\circ}) \iff f \in (q_r(\overline{\overset{\vee}{s^\circ}})) \to e^{-1}_{\sigma_1}(\overset{\vee}{t^\circ})$
but $((q_r(\overline{\overset{\vee}{s^\circ}})) \to e^{-1}_{\sigma_1}(\overset{\vee}{t^\circ}))$ is an open subset in the space of real functions in fact $(q_r(\overline{\overset{\vee}{s^\circ}}))$ is a compact set and $e^{-1}_{\sigma_1}(\overset{\vee}{t^\circ})$ is an open set.

vi) It remains to prove that: $q_{r\to\sigma_1} \circ e_{r\to\sigma_1} = id_{I\!R_{r\to\sigma_1}}$
Observe that for every function f in $I\!R_{r\to\sigma_1}$ and for every s in $I\!R$
$(q_{r\to\sigma_1} \circ e_{r\to\sigma_1})(f)(s) =$
$= q_{r\to\sigma_1}(e_{r\to\sigma_1}(f))(s) =$
$= (q_{\sigma_1} \circ (e_{r\to\sigma_1}(f)) \circ e_r)(s) =$
$= q_{\sigma_1}(e_{r\to\sigma_1}(f)(e_r(s))) =$ by point i)
$= q_{\sigma_1}(e_{\sigma_1} \circ f \circ q_r)(e_r)(s) =$
$= q_{\sigma_1} \circ e_{\sigma_1} \circ f \circ q_r \circ e_r)(s) =$ (by induction hypothesis)
$= f(s)$.
□

The functions e_σ and q_σ defined above are the natural generalizations of the functions e_r and q_r.

q_σ associates to each element of $(RD_\sigma)^\dagger$ the element of R_σ represented by it. e_σ chooses for each element in $I\!R_\sigma$ a canonical representation of it in $(RD_\sigma)^\dagger$.

We can also say that the function q_σ divides the elements of $RD^\dagger_\sigma$ in equivalence classes. All the elements contained in a single equivalence class represent the same element in $I\!R_\sigma$. The function e_σ defines a canonical representation for each class.

We discuss now the problem of defining an effective method that given an element f in $RD^\dagger_\sigma$, returns the canonical representation of the equivalence class of f. Such a method exists if the function $e_\sigma \circ q_\sigma : RD^\dagger_\sigma \to RD^\dagger_\sigma$ can be extended to a continuous and computable function $c_\sigma : RD_\sigma \to RD_\sigma$.

In the following we prove that such a function c_σ exists.

Notation. Given $s^\circ \in RD^\circ_\sigma$ we say that the finite set $I = \{s^\circ_1, \ldots, s^\circ_n | s^\circ_i \in RD^\circ_\sigma\}$ is a *partition covering* of s° if $\forall y \in \overline{\overset{\vee}{s^\circ}}.\ \exists s^\circ_i \in I.\ y \in \overset{\vee}{s^\circ_i}$.

We denote with $pc(s^\circ)$ the set of the partition coverings of s°.

Theorem 31 *For every type σ having rank strictly smaller that 2 ($\sigma = r$ or $\sigma \in T_1$) the function $e_\sigma \circ q_\sigma : RD^\dagger_\sigma \to RD^\dagger_\sigma$ can be extended to a continuous and computable function $c_\sigma : RD_\sigma \to RD_\sigma$, that is: $c_\sigma|_{RD^\dagger_\sigma} = e_\sigma \circ q_\sigma$.*
c_σ is the continuous extension of the function $c'_\sigma : RD^\circ_\sigma \to RD^\circ_\sigma$ defined by structural induction on the type σ as follows:

$c'_r([a,b]) = \bigsqcup\{[c,d] \mid c < a, b < d\}$

$$c'_{r\to\sigma_1}(g^\circ)(s^\circ) = \begin{cases} \bot & \text{if } s^\circ = \bot \\ \bigsqcup_{I\in pc(s^\circ)}\{\sqcap_{s'\in I}\{c'_\sigma \circ g^\circ \circ c'_r(s')\}\} & \text{otherwise} \end{cases}$$

Proof. It is easy to check that the c'_σ is a well defined and monotone function $RD^\circ_\sigma \to (RD^\circ_\sigma)$. It follows that c_σ is a well defined continuous function.

The prove of computability for c_σ is done by induction on σ. It is easy to show that the function c_r is computable. The computability of $c_{r\to\sigma_1}$ follows by induction hypothesis and by the fact that given a finite element $s^\circ \in RD_r$ it is possible to enumerate in an effective way the partition coverings of s°

In order to prove the equality $c_\sigma|_{RD^\dagger_\sigma} = e_\sigma \circ q_\sigma$ we first state the following two lemmas.

Lemma 32 *For every continuous function $g : RD_{r\to\sigma}$ and for every finite element $s^\circ \in RD^\circ_r$ we have:*

$$c_{r\to\sigma}(g)(s^\circ) = \begin{cases} \bot & \text{if } s^\circ = \bot \\ \bigsqcup_{I\in pc(s^\circ)}\{\sqcap_{s'\in I}\{c_\sigma \circ g \circ c_r(s')\}\} & \text{otherwise} \end{cases}$$

The proof is easy.

Lemma 33 *For every continuous function $g : RD_{r\to\sigma}$ and for every finite element $s^\circ \in RD^\circ$, $s^\circ \neq \bot$ we have:*

$$\bigsqcup\{t^\circ \mid g(\overset{\vee}{\overline{s^\circ}}) \subseteq \overset{\vee}{t^\circ}\} = \bigsqcup_{I\in pc(s^\circ)}\{\sqcap_{s'\in I}\{g(s')\}\}$$

Proof of the lemma.

We proof first that:

$$\bigsqcup\{t^\circ \mid g(\overset{\vee}{\overline{s^\circ}}) \subseteq \overset{\vee}{t^\circ}\} \sqsubseteq \bigsqcup_{I\in pc(s^\circ)}\{\sqcap_{s'\in I}\{g(s')\}\}$$

Let $t'^\circ \in RD^\circ_\sigma$ be a finite element such that: $t'^\circ \sqsubseteq \bigsqcup\{t^\circ \mid g(\overset{\vee}{\overline{s^\circ}}) \subseteq \overset{\vee}{t^\circ}\}$,
then: $g(\overset{\vee}{\overline{s^\circ}}) \subseteq \overset{\vee}{t'^\circ}$, that is: $\forall x \in \overset{\vee}{\overline{s^\circ}}.\ t'^\circ \sqsubseteq g(x)$.
By the continuity of g we have:
$\forall x \in \overset{\vee}{\overline{s^\circ}}.\ \exists x^\circ \in RD^\circ.\ (s^\circ \sqsubseteq x^\circ \sqsubseteq x) \wedge (t'^\circ \sqsubseteq g(x^\circ))$.

If $s^\circ \neq \bot$ then $\overline{\overset{\vee}{s^\circ}}$ is a compact set so we can find a finite set $I = \{x_1^\circ \dots x_n^\circ\}$ such that I is a partition covering of s° and $\forall i\ 1 \leq i \leq n.\ t'^\circ \sqsubseteq g(x_i^\circ)$
From this follows that: $t'^\circ \sqsubseteq \bigsqcup_{I \in pc(s^\circ)} \{\sqcap_{s' \in I} \{g(s')\}\}$
The inequality is thus proved.

Next we prove the inverse:

$$\bigsqcup_{I \in pc(s^\circ)} \{\sqcap_{s' \in I} \{g(s')\}\} \sqsubseteq \bigsqcup \{t^\circ \mid g(\overline{\overset{\vee}{s^\circ}}) \subseteq \overset{\vee}{t^\circ}\}$$

Let $t'^\circ \in RD_\sigma^\circ$ be such that:
$t'^\circ \sqsubseteq \bigsqcup_{I \in pc(s^\circ)} \{\sqcap_{s' \in I} \{g(s')\}\}$
This implies that there is a finite set $\{I_1 \dots I_n\}$ of partition coverings for s° such that:
$t'^\circ \sqsubseteq \bigsqcup_{1 \leq j \leq n} \{\sqcap_{s' \in I_j} \{g(s')\}\}$

Let d be the rational number
$d = \min\{(b-a) \mid \exists j.\ [a,b] \in I_j\}$
and consider the partial covering I of s° defined as
$I = \{[a,b] \mid b - a = d \text{ and } s^\circ \sqsubseteq [a,b]\}$
then: $\forall j.\ \sqcap_{s' \in I_j} \{g(s')\} \sqsubseteq \sqcap_{s'' \in I} \{g(s'')\}$ which yields:
$\bigsqcup_{1 \leq j \leq n} \{\sqcap_{s' \in I_j} \{g(s')\}\} \sqsubseteq \sqcap_{s'' \in I} \{g(s'')\}$ which implies:
$t'^\circ \sqsubseteq \sqcap_{s'' \in I} \{g(s'')\}$ which in turn:
$\forall s'' \in I.\ g(\overline{\overset{\vee}{s''}}) \subseteq \overset{\vee}{t'^\circ}$ and so $g(\overline{\overset{\vee}{s^\circ}}) \subseteq \overset{\vee}{t'^\circ}$
that is
$t'^\circ \sqsubseteq \bigsqcup \{t^\circ \mid g(\overline{\overset{\vee}{s^\circ}}) \subseteq \overset{\vee}{t^\circ}\}$.

This complete the proof of the lemma.

We can now prove by structural induction on c_σ that $c_\sigma|_{RD_\sigma^\dagger} = e_\sigma \circ q_\sigma$.

Basic step. Easy.

Inductive step. Given a finite element $s^\circ \in RD^\circ$ and $g \in RD^\dagger_{r \to \sigma}$
$(e_{r\to\sigma} \circ q_{r\to\sigma})(g)(s^\circ) = e_{r\to\sigma}(q_\sigma \circ g \circ e_r)(s^\circ) =$

$$= \begin{cases} \bot & \text{if } s^\circ = \bot \\ \bigsqcup \{t^\circ \mid e_\sigma \circ q_\sigma \circ g \circ e_r \circ q_r(\overline{\overset{\vee}{s^\circ}}) \subseteq \overset{\vee}{t^\circ}\} & \text{otherwise} \end{cases}$$

by induction hypothesis

$$= \begin{cases} \bot & \text{if } s^\circ = \bot \\ \bigsqcup \{t^\circ \mid c_\sigma \circ g \circ c_r(\overline{\overset{\vee}{s^\circ}}) \subseteq \overset{\vee}{t^\circ}\} & \text{otherwise} \end{cases}$$

from lemma 2

$$= \begin{cases} \bot & \text{if } s^\circ = \bot \\ \bigsqcup_{I \in pc(s^\circ)} \{\sqcap_{s' \in I} \{(c_\sigma \circ g \circ c_r)(s')\}\} & \text{otherwise} \end{cases}$$

and from lemma 1

$= c_{r\to\sigma}(g)(s')$.

□

The retract construction cannot be extended to spaces of functions having rank strictly larger than 1. In fact, given a type $\sigma \in T_2'$, although it is possible to define a function $q_\sigma : RD_\sigma^\dagger \to \mathbb{R}_\sigma$ that associates, in a natural way, to each element in $RD_\sigma^\dagger$ the function on real numbers represented by it, there is no topology on $\mathbb{R}_\sigma$ and no function $e_\sigma : \mathbb{R}_\sigma \to RD_\sigma^\dagger$ which makes the pair of functions q_σ, e_σ a retraction.

As far as the functions having rank 2 are concerned, we give the following result.

Theorem 34 *For every type σ of rank 2 there exist two functions $q_\sigma : RD_\sigma^\dagger \to \mathbb{R}_\sigma$ and $e_\sigma : \mathbb{R}_\sigma \to RD_\sigma^\dagger$ such that $q_\sigma \circ e_\sigma = id_{\mathbb{R}_\sigma}$.*

The functions are defined by structural induction as follows:
$q_{\sigma\to\sigma'} := \lambda G.\, q_{\sigma'} \circ G \circ e_\sigma$
$e_{\sigma\to\sigma'}(F)$ is the continuous extension of the function $\overline{F} : RD_\sigma^\circ \to RD_{\sigma'}$

$$\overline{F}(s^\circ) := \begin{cases} \bigsqcup\{t^\circ \mid e_{\sigma'} \circ F \circ q_\sigma(\overline{\overset{\vee}{s^\circ}}) \subseteq \overset{\vee}{t^\circ}\} & \text{if } \sigma = r \\ \bigsqcup\{t^\circ \mid e_{\sigma'} \circ F \circ q_\sigma(c_\sigma(s^\circ)) \subseteq \overset{\vee}{t^\circ}\} & \text{if } \sigma \in T_1 \end{cases}$$

where with $\overline{\overset{\vee}{c_\sigma(s^\circ)}}$ we denote the set $\{s \in RD_\sigma^\dagger \mid c_\sigma(s^\circ) \sqsubseteq s\}$

Proof. The proof is similar to the one given for Proposition 30. The main difference is the following. To prove that $e_{\sigma\to\sigma'}$ is well defined, it is necessary to prove that for every $\sigma \in T_1$ and for every $s^\circ \in RD_\sigma^\circ$ the set $\overline{\overset{\vee}{c_\sigma(s^\circ)}}$ is not empty.

The details of this proof are left to the reader.

□

Also in this case q_σ associates to each functional on RD the functional on $\mathbb{R}$ represented by it, and e_σ chooses a canonical representation for each continuous functional on $\mathbb{R}$. The fact that $q_\sigma \circ e_\sigma = id$ implies that all continuous functionals on $\mathbb{R}$ can be represented by an appropriate functional on RD.

A new definition of computability for functionals can now be given:

Definition 17 *Given a type σ with $\partial(\sigma) \leq 2$, an element $f \in \mathbb{R}_\sigma$ is* RD-computable *if there exists a computable element $g \in RD_\sigma$ such that $f = q_\sigma(g)$.*

It is interesting to observe that the definitions of RD-computable and of domain-computable function coincide:

Proposition 35 *For every type $\sigma \in T'$ with $\partial(\sigma) \leq 2$ the sets $\mathbb{R}_\sigma$ and $\mathbb{R}_\sigma^d$ coincide and both contain the same subset of computable elements.*

Proof Follows immidiately from Theorem 34.

From the above a very interesting property of computable functionals it follows.

Theorem 36 *Every RD-computable functional is continuous, w.r.t. the compact open topology on function spaces.*

Proof Omitted.

This is a useful criteria for determining the non computability of functionals. Starting from this result it is easy to demonstrate that the following functionals are not computable.

1. Derivative.
$$F(f) = \frac{df}{dx}$$
2. The functional that given a function f and an interval [a,b] yields the point x in $[a, b]$ where the value f(x) is minimum.
$$F(f, [a, b]) = x \text{ if } f(x) = \min\{f(y) \mid y \in [a, b]\}$$
3. The functional that given a function f and an interval $[a, b]$ yields a point x in $[a, b]$ where the value $f(x)$ is zero or is equal to a if such a value does not exist.
$$F(f, [a, b]) = \begin{cases} x & \text{if } f(x) = 0 \text{ and } x \in [a, b] \\ a & \text{if } \forall x \in [a, b].\ f(x) \neq 0 \end{cases}$$

So we have thus an important topological characterization of the computable functionals on the real numbers. We are not aware of any similar result in the literature.

6.3 Partial functions

It is possible to extend the interpretation functions q_σ to the whole space RD_σ.

An element in RD_σ, not contained in $RD^\dagger_\sigma$, denotes a partial function on $\mathbb{R}$.

In the following we give a topological characterisation of the partial real functions obtained in this way.

Definition 18 *Given a type $\sigma = \sigma_1 \to \ldots \to \sigma_n \to r$ such that: $1 \leq \partial(\sigma) \leq 2$, $\mathbb{R}^p_\sigma$ denotes the set of partial functions from $\mathbb{R}_{\sigma_1} \times \ldots \times \mathbb{R}_{\sigma_n}$ to $\mathbb{R}$.*
The function *$p_\sigma : RD_\sigma \to \mathbb{R}^p_\sigma$ is defined by:*
$$p_r(s) := \begin{cases} q_r(s) & \text{if } s \in RD^\dagger \\ \text{undefined} & \text{otherwise} \end{cases}$$
$$p_\sigma(g) := \lambda\langle x_1 \ldots x_n\rangle.p_r(\ldots(g(e_{\sigma_1}(x_1)))\ldots(e_{\sigma_n}(x_n))))$$

In the definition of $\mathbb{IR}^p_\sigma$ an operation of "currying" has been carried out. Using this trick the function p_σ can be interpreted as an extension of the function q_σ.

Proposition 37 *Given a type σ with $\partial(\sigma) = 1$, the partial function $h \in \mathbb{IR}^p_\sigma$ belongs to the image of the function p_σ if and only if*
1) the domain of h is a countable intersection of open sets.
2) h is continuous on its domain.

Proof Omitted.

It is now natural to extend the notion of RD-computable function to the partial functions

Definition 19 *An element h in $\mathbb{IR}^p_\sigma$ is RD-computable in there is a computable element g in RD_σ such that $h = p_\sigma(g)$.*

So we have that every RD-computable partial function h is defined on a set that is countable intersection of open sets and is continuous on its domain of definition.

Chapter 7

A programming language for real numbers

In chapters 3 and 4 we showed how computable real numbers and computable functions on real numbers can be defined via closed PCFS terms.

This was done either using the partial interpretation functions $R^i[\![\ ]\!]_\sigma$ or using the partial functions $\mathrm{va}^i_\sigma(E[\![\ ]\!])$. These functions however can not be total functions.

Let us consider for example the digit notation and corresponding functions $R^3[\![\ ]\!]_\sigma$. Given a closed term $f \in E_{S(N)\to S(N)}$ the value $R^3[\![f]\!]_{r\to r}$ can be undefined for various reasons. Various phenomena can in fact occur
i) there can be a term e representing a real number such that the evaluation of $f\,e$ is undefined;
ii) there can be a term e representing a real number such that the evaluation of $f\,e$ defines a sequence containing not admissible digits.
iii) there can be two terms e_1 and e_2 defining the same real number and such that fe_1 and fe_2 define different real numbers.

We investigate now the possibility of defining a language for computation on real numbers which is free from at least some of the undesired phenomena discussed above. In particular we will show how to avoid inadequacies of the kind ii) and iii). We should look for a language which generates just correct sequence of digits, but more importantly it preserve the equivalence between different representation of the same real number.

Moreover we require that the language is sufficiently expressive to define all computable real functions.

Equivalently we can express our task also as the problem of finding a definition of computable real numbers as an abstract date type. This amounts to finding a collection of primitive functions on reals which generate all other computable functions on real numbers.

The solution we propose is given utilizing the domain RD. We present a simply typed lambda calculus called RL. RL has a set of constants denoting a set of computable functions on RD. Any other computable function on RD will then be denotable by a suitable term in RL.

In this way, each term e having type r denotes a real or is a partial approximation of a real, and each term f having type σ with $\partial(\sigma) \leq 2$ denotes a (possibly partial) function on real numbers. All partial computable functions as defined in the previous chapter can be denoted in this way.

7.1 Syntax

The set of type of RL is the set T' defined in chapter 3.

T' is generated by the grammar:

$$\sigma := r \mid \sigma \to \sigma$$

The terms of RL are:

$$e \ := \ x^\sigma \mid c^\sigma \mid (e^{\sigma \to \sigma'})e^\sigma \mid \lambda x^\sigma . e^{\sigma'}$$

where c^σ is a metavariable ranging over the set of constants C whose elements are:

$$(-1), \ \ (+1), \ \ (\times 2), \ \ (\div 2), \ \ PR \ : \ r \to r,$$
$$\text{cond} \ : \ r \to r \to r \to r, \ \ \exists \ : \ (r \to r) \to r$$
$$Y_\sigma \ : \ (\sigma \to \sigma) \to \sigma$$

type assignments and type constraints are defined as usual.

7.2 Semantics

The denotational semantics for RL is given using the set of domains
$UD := \{RD_\sigma \mid \sigma \in T'\}$

The semantic interpretation function E has the form:

$$E : RL \to Env \to UD$$

where Env is the set of environments. An environment is a function ρ from Var to UD satisfying the condition

$$\rho(x^\sigma) \in RD_\sigma$$

The definition of E is given by structural induction,

$$\begin{array}{lcl} E[\![c]\!]_\rho & := & B[\![c]\!] \\ E[\![x^\sigma]\!]_\rho & := & \rho(x^\sigma) \\ E[\![e^{\sigma \to \tau} e^\sigma]\!]_\rho & := & E[\![e^{\sigma \to \tau}]\!]_\rho (E[\![e^\sigma]\!]_\rho) \\ E[\![\lambda x^\sigma . e^\tau]\!]_\rho & := & \lambda a \in RD_\sigma . E[\![e^\tau]\!]_{(\rho[a/x])} \end{array}$$

The interpretation of constants is defined using the function B :

The constants $(+1), (-1), (\times 2), (\div 2)$ implement the obvious functions on the intervals (definition 12) viz.

$$B[\![(+1)]\!], B[\![(-1)]\!], B[\![(\times 2)]\!], B[\![(\div 2)]\!] : RD \to RD$$

are the continuous extension of the functions

$$B^\circ[\![(+1)]\!], B^\circ[\![(-1)]\!], B^\circ[\![(\times 2)]\!], B^\circ[\![(\div 2)]\!] : RD^\circ \to RD$$

$$\begin{array}{lcl} B^\circ[\![(+1)]\!]([a,b]) & = & [a,b]+1 \\ B^\circ[\![(-1)]\!]([a,b]) & = & [a,b]-1 \\ B^\circ[\![(\times 2)]\!]([a,b]) & = & \begin{cases} [-\infty,+\infty] & \text{if } b-a \geq 2 \\ [a,b]\times 2 & \text{otherwise} \end{cases} \\ B^\circ[\![(\div 2)]\!]([a,b]) & = & [a,b] \div 2 \end{array}$$

The constant PR implements a kind of projection on the interval $[-1,1]$. If $[a,b]$ is consistent with $[-1,1]$ then $B[\![PR]\!]([a,b])$ is equal to $[a,b] \sqcup [-1,1]$. Formally $B[\![PR]\!]$ is the continuous extension of the function $B^\circ[\![PR]\!] : RD^\circ \to RD$,

$$B^\circ[\![PR]\!]([a,b]) := \begin{cases} [a,b] & if & -1 \leq [a,b] \leq 1 \\ \overline{1^-} & if & 1 \leq [a,b] \\ \overline{-1^+} & if & [a,b] \leq -1 \\ [\,\max\{-1,a\}, \min\{1,b\}\,] & \text{otherwise} & \end{cases}$$

The constant cond is a test function. $B[\![\text{cond}]\!]$ checks if its first argument is smaller or bigger than zero. $B[\![\text{cond}]\!]$ is the continuous extension of the function

$$B^\circ[\![\text{cond}]\!] : RD^\circ \to (RD^\circ \to (RD^\circ \to RD))$$

$$B^\circ[\![\text{cond}]\!]([a,b])([c,d])([e,f]) := \begin{cases} [c,d] & if & [a,b] \leq 0 \\ [e,f] & if & 0 \leq [a,b] \\ [c,d] \sqcap [e,f] & \text{otherwise} & \end{cases}$$

If the precision of the first argument is not sufficient to decide if it is smaller or bigger than zero, the function $B[\![\text{cond}]\!]$ gives as output the most precise approximation of the second and third argument.

Remark. The function cond is a parallel test function and cannot be implemented sequentially. In [8] there is a proof showing that sequential primitives are not sufficient to define real numbers as an abstract data type.

The constant $\exists$ is similar to the corresponding constant introduced in LCF by Plotkin [21].

In LCF the interpretation of $\exists$ is the function $H : (Nat \to Bool) \to Bool$ defined by:

$$H(f) := \begin{cases} false & if \;\; f(\bot) = false \\ true & if \;\; \exists n. f(n) = true \\ \bot & otherwise \end{cases}$$

$\exists$ checks if there is any n making $f(n)$ true.

In our case the constant $\exists$ checks if a function $f : RD \to RD$ gives for some input value a result that is an approximation of a value smaller than zero.

The definition for $B[\![\exists]\!]$ is

$$B[\![\exists]\!](g) := \begin{cases} \overline{1} & if \;\; g(\bot) = [a,b] \;\wedge\; 0 < a \\ \overline{-1} & if \;\; \exists [c,d].\, g([c,d]) = [a,b] \;\wedge\; b < 0 \\ \bot & \text{otherwise} \end{cases}$$

The constants $Y^{(\sigma\to\sigma)\to\sigma}$ are the usual fixed point operators.

The following theorems relate computable elements in RD and closed expressions in RL.

Proposition 38 *For every closed expression e^σ and environment ρ, $E[\![e^\sigma]\!]_\rho$ is a computable element of RD_σ.*

Proof. It is straightforward to prove that every expression in RL denotes a computable element if and only if every constant in RL denotes a computable element. To prove this second fact we use the following characterization of the computable elements of a Scott Domain: an element in a Scott domain is computable if it is the lub of a primitive recursive set of finite elements.

The proof follows in a straightforward way from the well known fact that the recursive operators Y^σ are computable and from the following equalities:

$$B[\![(+1)]\!] = \bigsqcup en_r(\langle m, m', n\rangle + 1) \Rightarrow en_r(\langle m+1, m', n\rangle + 1)$$

$$B[\![(-1)]\!] = \bigsqcup en_r(\langle m, m', n\rangle + 1) \Rightarrow en_r(\langle m, m'-1, n\rangle + 1)$$

$$B[\![(\div 2)]\!] = \bigsqcup en_r(\langle m, m', n\rangle + 1) \Rightarrow en_r(\langle m, m', n+1\rangle + 1)$$

$$B[\![(\times 2)]\!] = \bigsqcup en_r(\langle m, m', n+1\rangle + 1) \Rightarrow en_r(\langle m, m', n\rangle + 1)$$

$$\begin{aligned} B[\![\text{cond}]\!] = \;& \bigsqcup en_r(\langle 0, m'+1, 0\rangle + 1) \Rightarrow en_r(p) \Rightarrow en_r(0) \Rightarrow en_r(p) \\ & \sqcup \bigsqcup en_r(\langle m+1, 0, 0\rangle + 1) \Rightarrow en_r(0) \Rightarrow en_r(p) \Rightarrow en_r(p) \\ & \sqcup \bigsqcup en_r(0) \Rightarrow en_r(p) \Rightarrow en_r(p) \Rightarrow en_r(p) \end{aligned}$$

$$\begin{aligned} B[\![\exists]\!] = \;& \bigsqcup (en_r(p) \Rightarrow en_r(\langle 0, 2^n + m' + 2, n\rangle + 1)) \Rightarrow en_r(\langle 0, 2^{n'}, n'\rangle + 1) \\ & \sqcup \bigsqcup (en_r(0) \Rightarrow en_r(\langle 2^n + m + 2, 0, n\rangle + 1)) \Rightarrow en_r(\langle 0, 2^{n'}, n'\rangle + 1) \end{aligned}$$

$$B[\![PR]\!] = \bigsqcup_{|m-m'|\leq 2^n} en_r(\langle m, m', n\rangle + 1) \Rightarrow en_r(\langle m, m', n\rangle + 1)$$
$$\sqcup \bigsqcup en_r(\langle m + 2, 0, 0\rangle + 1) \Rightarrow en_r(\langle 2^n - 1, 0, n\rangle + 1)$$
$$\sqcup \bigsqcup en_r(\langle 0, m' + 2, 0\rangle + 1) \Rightarrow en_r(\langle 0, 2^n - 1, n\rangle + 1)$$
$$\sqcup en_r(0) \Rightarrow en_r(\langle 0, 0, 0\rangle + 1)$$

□

Theorem 39 *For every computable element d in RD_σ there exists a closed expression e^σ in RL such that: $E[\![e^\sigma]\!]_\rho = d$.*

Proof. The essential idea in this proof is taken from the proof of theorem 5.1 in [21].

In the proof it is necessary to use the primitive recursive functions. In order to denote function on natural number by RL terms, we code the natural numbers using elements of RD. We choose to code the natural number n by the element $\overline{n}$.

We say that the function $\overline{f}$ in RD represents the function f between natural numbers if

$$\overline{f}(\overline{n}_1)\ldots(\overline{n}_i) = \overline{n} \text{ whenever } f(n_1 \ldots n_i) = n.$$

We say that a closed term t in RL defines a function f on natural numbers if $E[\![t]\!]_\perp$ represents f.

Two lemmas are necessary here.

Lemma 40 *For every primitive recursive function f there is a closed term f in RL which defines f.*

Proof. Left to the reader. Observe that the constant 0 is defined by the term $k_0 = Y_r(\lambda f.\lambda x.(\div 2)(PR\ x))$. The function successor is defined by the constant $(+1)$.

A second step in the proof is to show that for every type σ it is possible to define three functions C_σ, P_σ and $\#_\sigma$. Where C_σ and P_σ are respectively a test function and a projection function for high order types, while $\#_\sigma(\overline{n})(\alpha^\sigma)$ checks if the element α^σ is greater than the finite element $en_\sigma(n)$. More formally:

Lemma 41 *For every type σ there exist three terms C_σ, P_σ and $\#_\sigma$ satisfying the following conditions:*

C_σ has type $r \to \sigma \to \sigma \to \sigma$ and

$$E[\![C_\sigma]\!]([a,b])(\alpha_1^\sigma)(\alpha_2^\sigma) = \begin{cases} \alpha_1^\sigma & if \quad b \leq 0 \\ \alpha_2^\sigma & if \quad 0 \leq a \\ \alpha_1^\sigma \sqcap \alpha_2^\sigma & if \quad a < 0 < b \end{cases}$$

P_σ has type $r \to \sigma \to \sigma$ and if $en_\sigma(n)$ and α^σ are consistent then $E[\![P_\sigma]\!](\overline{n})(\alpha^\sigma)$ is equal to $en_\sigma(n) \sqcup \alpha^\sigma$.

$\#_\sigma$ *has type* $r \to \sigma \to r$ *and*

$$E[\![\#_\sigma]\!](\overline{n})(\alpha^\sigma) = \begin{cases} \overline{1} & \text{if } e^\sigma(n) \sqsubseteq \alpha^\sigma \\ \overline{-1} & \text{if } e^\sigma(n) \text{ and } \alpha^\sigma \text{ are inconsistent} \\ \bot & \text{otherwise} \end{cases}$$

Proof. In the course of the proof we need to use the following terms:

$$\Omega_\sigma = Y_\sigma(\lambda\alpha^\sigma.\alpha^\sigma)$$

$$Z = \lambda n.(-1)((\times 2)n)$$

$$k_0 = Y_r(\lambda f.\lambda x.(\div 2)(PR\ x)) \quad k_1 = (+1)k_0 \quad k_{-1} = (-1)k_0$$

$$plus = \lambda x.Y_{r\to r}(\lambda f.\lambda n.\text{if}_r\ Z\ n \text{ then } x \text{ else } ((+1)(f((-1)n))))$$

$$minus = \lambda x.Y_{r\to r}(\lambda f.\lambda n.\text{if}_r\ Z\ n \text{ then } x \text{ else } ((-1)(f((-1)n))))$$

$$div = \lambda x.Y_{r\to r}(\lambda f.\lambda n.\text{if}_r\ Z\ n \text{ then } x \text{ else } ((\div 2)(f((-1)n))))$$

$$times = \lambda x.Y_{r\to r}(\lambda f.\lambda n.\text{if}_r\ Z\ n \text{ then } x \text{ else } ((\times 2)(f((-1)n))))$$

$E[\![\Omega_\sigma]\!]$ is the bottom element in RD^σ.
When n is equal to zero $E[\![Z\,]\!](\overline{n})$ is a negative value, for n natural number different from zero $E[\![Z\,]\!](\overline{n})$ is a positive value. $E[\![k_0]\!]$, $E[\![k_1]\!]$, $E[\![k_{-1}]\!]$ are the values $\overline{0}$, $\overline{1}$, $\overline{-1}$.
For every natural number n $E[\![plus\,]\!]([a,b])(\overline{n})$, $E[\![minus\,]\!]([a,b])(\overline{n})$, $E[\![times\,]\!]([a,b])(\overline{n})$ and $E[\![div\,]\!]([a,b])(\overline{n})$ are the intervals $[a+n, b+n]$, $[a-n, b-n]$, $[a\times 2^n, b\times 2^n]$ and $[a/2^n, b/2^n]$ respectively.

Moreover we have by lemma 40 *DIV* , *MOD* and *EXP* indicate *RL* terms defining the primitive recursive functions f, g and h such that $h(n)$ is equal to 2^n, $f(m,n)$ is the result of the rounded division of m by 2^n and $g(m,n)$ is the rest of the division of m by 2^n. In other words $m = f(m,n)\times 2^n + g(m,n)$ and $g(m,n) < 2^n$.

Again by lemma 40 we have the terms Π_1, Π_2, Π_3 which define three primitive recursive functions i_1, i_2, i_3 such that:

$\forall n \neq 0.\ n = \langle i_1(n), i_2(n), i_3(n)\rangle + 1$ that is
$\forall n \neq 0.\ en_r(n) = [(i_1(n) - i_2(n) - 1)/2^{i_3(n)}\,,\ (i_1(n) - i_2(n) + 1)/2^{i_3(n)}]$

The proof is by induction on the type σ.

Basic step:

The term C_r is simply the constant cond.
In the following $(\text{if}_\sigma\ t_1 \text{ then } t_2 \text{else } t_3)$ will stand for $C_\sigma(t_1)(t_2)(t_3)$

The term P_r is:

$P_r = \lambda n.\lambda x.\text{if}_r\ Z\ n \text{ then } x \text{ else}$
$div\ (plus\ (minus\ (PR\ (plus\ (minus\ (times\ \ x(\Pi_3 n))(\Pi_1 n))(\Pi_2 n)))(\Pi_2 n))(\Pi_1 n))(\Pi_3 n)$

To define the term $\#_r$ we first define an auxiliary term *out*:

$$\begin{array}{ll} out = & Y_{r\to r\to r\to r}\lambda f.\lambda m.\lambda n.\lambda x. \\ & \text{if}_r \;\; minus\; x\,(DIV\; m\; n) \\ & \quad \text{then}\, k_{-1} \\ & \quad \text{else if}_r \;\; minus\; x(DIV\; (plus\; m\; ((+1)(EXP\; n)))\; n) \\ & \qquad \text{then if}_r\; Z\; n \;\; \text{then}\, k_1 \\ & \qquad\qquad \text{else}\, f(MOD\; m\; n)((-1)n)(minus\; x\; (DIV\; m\; n)) \\ & \qquad \text{else}\, k_{-1} \end{array}$$

The term *out* has the following behavior:

$$E[\![out]\!](\overline{m})(\overline{n})(x) = \begin{cases} \overline{1} & \text{if } [m/2^n, (m+2)/2^n] \sqsubseteq x \\ \overline{-1} & \text{if } [m/2^n, (m+2)/2^n] \text{ and } x \text{ are inconsistent} \\ \bot & \text{otherwise} \end{cases}$$

Finally we can put:

$$\begin{array}{ll} \#_r = & \lambda n.\lambda x.out(plus\; (\Pi_1 n)(minus\; (EXP\; (\Pi_3 n))(MOD\; (\Pi_2 n)(\Pi_3 n)))) \\ & (\Pi_3 n)\; (plus\;\; x\; ((+1)(DIV\; (\Pi_2 n)(\Pi_3 n))) \end{array}$$

Inductive step:

let $\sigma = \sigma_1 \to \sigma_2$.

The definition of C_σ and P_σ are the following ones:

$$C_\sigma = \lambda x.\lambda\alpha_1^\sigma.\lambda\alpha_2^\sigma.\lambda\beta^{\sigma_1}.\text{if}_{\sigma_2} x \text{ then } \alpha_1^\sigma\beta^{\sigma_1} \text{ else } \alpha_2^\sigma\beta^{\sigma_1}$$

$$\begin{array}{ll} P_\sigma = & \lambda m.\lambda\alpha^\sigma.\lambda\beta^{\sigma_1}. \\ & Y_{r\to\sigma}(\lambda f.\lambda n.\text{if}_{\sigma_2} Z\; n \;\; \text{then}\, \alpha^\sigma\beta^{\sigma_1} \\ & \qquad\qquad \text{else if}_{\sigma_2} \;\; \#_{\sigma_1}(FIRST_\sigma\;\; m\, n)\beta^{\sigma_1} \\ & \qquad\qquad\qquad\qquad \text{then}\, f((-1)n) \\ & \qquad\qquad\qquad\qquad \text{else}\, P_{\sigma_2}(SECOND_\sigma\;\; m\, n)(f((-1)n))) \\ & (SIZE_\sigma\; m) \end{array}$$

Where $FIRST_\sigma$, $SECOND_\sigma$ and $SIZE_\sigma$ define three primitive recursive functions f, g and h such that

$$en_\sigma(m) = \bigsqcup\{en_{\sigma_1}(f(m,n)) \Rightarrow en_{\sigma_2}(g(m,n)) \mid 0 < n \leq h(m)\}$$

We introduce other two auxiliary terms: *norm* and $\overline{\exists}$. Their definitions are:

$$norm = Y_{r\to r}\lambda f.\lambda x.\text{if}_r Z\; x \text{ then } k_0 \text{ else } (+1)(f((-1)x)))$$

$E[\![norm]\!]$ is a function that transforms every input value in bottom or in a value of the form $\overline{n}$ for same natural number n. Moreover for every natural number, n $E[\![norm]\!](\overline{n}) = \overline{n}$.

$$\overline{\exists} = \lambda f.(\exists(\lambda x.f(norm\; x)))$$

$E[\![\overline{\exists}]\!]g$ checks if there exists a natural number n such that $g(\overline{n})$ is a smaller than zero in the real line order.

Finally the term $\#_\sigma$ is defined by:

$$\begin{array}{ll}\#_\sigma = & \lambda m.\lambda\alpha^\sigma.Y^{r\to r}(\lambda f.\lambda n.\\ & \text{if}_r\ \ Z\,n\\ & \quad \text{then}\, k_1\\ & \quad \text{else if}_r\ \ \overline{\exists}(\lambda l.\#_{\sigma^2}(SECOND_\sigma\, m\, n)(\alpha^\sigma(P_{\sigma_1}(FIRST_\sigma\, m\, n)(P_{\sigma_1}\, l\, \Omega_{\sigma_1}))))\\ & \quad\quad \text{then}\, k_{-1}\\ & \quad\quad \text{else}\ \ f((-1)n))\\ & (SIZE_\sigma\ m)\end{array}$$

Observe that the function $E[\![\lambda l.P_\sigma n(P_\sigma\, l\, \Omega_\sigma)]\!]$ enumerates the finite elements in the set $\{x^\circ \mid en_\sigma(n) \sqsubseteq x^\circ\}$, and $P_\sigma n(P_\sigma\, \Omega_r\, \Omega_\sigma) = en_\sigma(n)$.

This ends the proof of the lemma.

It is now possible to prove the theorem. Given a computable element x in RD^σ let f be a primitive recursive function such that

$$x = \bigsqcup\{en_\sigma(f(i)) \mid i \in N\}$$

and let F be a term defining the function f then we have:

$$x = E[\![Y_{r\to\sigma}(\lambda g.\lambda n.P_\sigma(Fn)(g((+1)n)))k_0]\!]$$

□

In RL it is still possible to distinguish between different representations of a given real number. In fact RL is a language based on the domain RD where every dyadic rational have three representations. As a consequence in RL it is possible to denote functions distinguishing between these three representations. An example is the term
$\lambda x^r.\text{cond}_r x k_0 k_1$
where $k_0 = Y_r(\lambda f.\lambda x.(\div 2)(PR\ x)) \quad k_1 = (+1)k_0$
which discriminates between the three RD representation of the number 0.

There is however an simple way to construct only "consistent" functions. In proposition 31 the computable function c_r is defined. c_r sends evey representation of a real number in a canonical representation of the same number. c_r has been proved to be computable therefore there is a RL term C_r denoting it. It is straightforward to prove that if e_f is a closed term then the term $\overline{e_f} = \lambda x^r.e_f(C_r x^r)$ denotes the same function:
$p_{r\to r}(E[\![e_f]\!]_\perp) = p_{r\to r}(E[\![\overline{e_f}]\!]_\perp)$
and moreover $\overline{e_f}$ respects the equivalence relation between different representations of a real number.

The above construction can be easily generalized to terms denoting functions with several arguments.

Chapter 8

Implementation Issues.

In this chapter we investigate the issue of efficiency of the exact real number computation.

The implementation of the arithmetic operations play an important role in the efficiency of computation on real number. In fact arithmetic operations are the basic functions in most of the ordinary computations on real numbers. For example the evaluation of an analytic function is almost always reduced to the evaluation of a series. In the following we develop real number representations that can lead to efficient implementation of arithmetic functions.

Surprisingly the literature on the efficiency of the real computation is very small. Important contributions to this area are [6] and [33].

In [33] the continuous fraction notation is studied. That notation has been shown to be feasible for the real number computation. Efficient algorithms for a wide class of analytic functions are presented there.

Two other notations for real numbers, Cauchy sequences and digit notation, are considered instead in [6]. There it is shown that in several implementations the digit notation is the least efficient between the two.

This fact is rather surprising since from an abstract point of view the digit notation has some advantages. For instance, in the digit notation, in order to improve the accuracy, i.e. to increase the approximation, with which a result is evaluated it is possible to take advantage from the previous computation. This is not true for the Cauchy sequence notation. Here in order to improve the accuracy of an evaluation it is necessary to repeat (with an higher precision) almost all the computation.

Implementations based on the Cauchy sequence notation however lead to more efficient algorithms because this notation permits to exploit better the fact that integer arithmetic operations are always evaluated efficiently in modern computers.

In this chapter we intend to make a case for the use of digit notation to implement Reals computation. We propose two variants of the digit notation

that can improve the efficiency of implementations.

The two variants are designed so as to achieve efficient computations according to two different strategies.

The two strategies are:

1) to implement the algorithms for real arithmetic operations directly in the hardware.

2) to exploit the fact that in every computer, integer arithmetic is already efficiently implemented into the hardware.

8.1 A binary notation for Reals.

If we intend to implement arithmetic operations directly into the hardware it is useful to utilize digit notations that lead to a simplification of the complexity of the hardware needed.

The most widely used hardware notation for integers is the binary one. The question arises as to whether there are "two digit" notations suitable for real number computation. It turns out that such notations exist, and that there are infinitely many choices. We single out, among these, one that we claim to be optimal for hardware implementation, w.r.t. the simplicity of the algorithms.

8.2 Real-base notations.

In the second chapter we showed that the standard binary notation is not suitable for real number computation. Other forms of notation are therefore needed.

A preliminary remark here is appropriate. By the word "base" of a digit notation we mean the ratio between the weights of two consecutive digits in the notation. More explicitly: if the base is b, the finite string of digits $0.a_1 \dots a_n$ denotes the number $\sum_{1 \leq i \leq n} a_i/b^i$. Normally base and number of digits in a notation coincide. The distinction between the two concepts is therefore often blurred. But nothing prevents us from defining notations where base and number of digits are different, like in the negative digit notation. And nothing prevents us from choosing as base a number that is not natural. *Rational* or even *irrational* numbers can legitimately be chosen as bases. The only restriction is that the base has to be a number strictly larger than 1 and smaller than the number of digits in the notation.

Exploiting this idea it is possible to define a collection of real number notations.

Definition 20 *For every natural number n and real computable number b with $1 < b \leq n$, the* digit notation having base b and n digits *is defined as follows: a real number r is denoted by a triple $[z, l, s]$ where z is an integer, l is a finite sequence of natural numbers and s is an infinite sequence of natural numbers such that:*

i) $z \in \{-1, 1\}$
ii) $\forall i \in dom(l).\, 0 \leq l_i < n$
iii) $\forall i \in I\!N^+.\, 0 \leq s_i < n$
iv) $r = (z \times \sum_{i \in dom(l)} l_i \times b^i) + \sum_{i \in I\!N^+} s_i \times b^{-i}$

A notation for reals of the above form will be called a *Real-base notation*. Loosely speaking we can say that in the triple $[z, l, s]$ z represents the sign, l represents the "integral" part and s represents the "fractional" part. Note that the sign affects only the value denoted by the integer part.

The term $[z, l, s]$ can be also written in a form that is more reminiscent of the conventional "dot" notation. We will often indicate a triple $[+1, l, s]$ with the sequence $l'.s$ and a triple $[-1, l, s]$ with the sequence $-\,l'.s$ where l' is the finite sequence l written in the reverse order.

It is easy to prove that in any real-base notation, every real number can be denoted by a suitable triple, of corse not uniquely.

The real-base notation can be used to represent real numbers by PCFS terms. We have:

Definition 21 *Given a natural number n and a computable real number b with $1 < b < n$, the partial representation function $R^{4-n-b} : E_{S(N)} \to I\!R$ is defined by:*
$R^{4-n-b}[\![e]\!] = r$ *if there is a sequence of natural numbers $l_0, l_1, \ldots$ such that:*
i) $\forall i.car(cdr^i e) \Rightarrow k_{l_i}$
ii) $l_0 \in \{-1, 1\}$
iii) $\forall i \geq 2.\, 0 \leq l_i \leq n$
iv) $r = (l_0 \times \sum_{i=0}^{l_1} l_{(i+2)} \times b^i) + \sum_{i \in I\!N^+} l_{(i+l_1+2)} \times b^{-i}$

Informally: l_0 is the sign, the value l_1 is used to encode separation the sequence $l_2 : l_3 : \ldots$ in an integer part and in a fractional part.

For any suitable n and b the partial representation functions R^{4-n-b} is computationally equivalent to the partial representation functions presented in definition 5.

Proposition 42 *For every natural number n and computable real number b with $1 < b < n$, there are two PCFS terms $conv_{4-3}$ and $conv_{3-4}$ such that for every closed term $e \in E_{S(N)}$ the following equalities hold:*
$R^3[\![e]\!] \simeq R^{4-n-b}[\![conv_{4-3}\ e]\!]$
$R^{4-n-b}[\![e]\!] \simeq R^3[\![conv_{3-4}\ e]\!]$

Proof A slight modification of the standard algorithm for base conversion can be used for translating effectively from the negative digit notation to the real-base digit notation, and vice versa. The terms conv_{4-3} and conv_{3-4} are obtained by implementing inside PCFS these algorithms.
□

Using the machinery presented in chapters 3 and 4, definitions 8 and 10, it is possible to define, for each partial function R^{4-n-b}, a full hierarchy of both language-computable and domain-computable functions. By propositions 6 and 12, these sets of computable functions are equal to the sets defined in definitions 8 and 11.

Remark The idea behind the real-base notation can be pursued further. If the base of a digit notation is an imaginary number then a string of digit represents a complex number. Every complex number can be then represented in a digit representation, having a suitable imaginary number as base and a suitable number of digits. Algorithms for the the arithmetic operations on complex numbers can then be easily developed for these kinds of notations.

8.3 The golden ratio notation.

From the above results it follows that there are infinitely many binary digit notations suitable for real number computation. We discuss now in detail the one that we consider the most convenient for the simplicity of the algorithms.

It is worthwhile to mention that Brouwer was probably the first to notice the computational difficulties of the standard binary digit notation. He suggested as a possible solution the use of a binary notation with base 3/2 [3].

The binary notation with base 3/2 is not such a convenient choice if we want to obtain simple algorithms. See below for more details. We claim that the simplest algorithms for arithmetic operations are obtained when the value of the base is the golden ratio, that is the number $\varphi = (\sqrt{5}+1)/2$.

An intuitive motivation for this fact is the following: when the base is the golden ratio, the string 0.11 denotes the number 1. As a consequence, in the golden ratio notation, it is possible to rewrite a group of digits in a string without altering the value denoted by it. For example all the following strings denote the number φ^3, "100.0", "11.0", "10.11", "10.1011". Exploiting identities of this kind it is possible to obtain quite simple algorithms for the arithmetic operations.

The choice of the golden ratio as base is derived by identifying the values denoted by the two strings "1.00" and "0.11". The golden ratio is the only suitable value for a base, which leads to that identity.

The key idea here is that there is a strict correspondence between bases and rewrite rules for strings of digits which preserve the denotation of strings.

In fact other identities between values denoted by binary strings lead to other possible choice for the bases. For example if we identify the values denoted by 1000.0 and 111.0 the base would be different. The base 3/2 is obtained if we impose the identity $(1.0 + 1.0 + 1.0) = (10.0 + 10.0)$.

Also these identities can be used to derive algorithms for the arithmetic operation. But since the groups of digits involved in these identities are larger

than in the case of the golden ratio, the algorithms for the arithmetic operations turns out to be more complex.

8.4 Algorithms for Arithmetic Operations.

In the following we describe the algorithms in the case of the arithmetic operations for the golden ratio notation.

The algorithms are given in two steps. First we define the algorithms for a simplified notation of a significant subset of the real numbers. The algorithms for the full notation can be then derived from the previous ones.

In the simplified notation a real number r is denoted by a sequence of binary digits $\alpha = \alpha_1 : \alpha_2 : \ldots$ such that:

$$r = \sum_{i \in I\!N^+} \alpha_i \times \varphi^{-i}$$

Just a subset of the real numbers can be denoted using this simplified notation. Namely the real numbers contained in the interval $[0, \varphi]$. This simplified notation is essentially the fractional part of the full notation.

After having defined the algorithms for the arithmetic functions in the simplified notation, it is not difficult to obtain the correspondent algorithms for the full notation. It is only necessary to take care of the sign and of the integral part. In the following we present just the algorithms for the simplified notation.

Since only a finite interval of Reals is represented in the simplified notation, an overflow problem arises. To avoid this problem we describe algorithms that yield results of the arithmetic operations divided by a power of φ. Note that in the golden ratio notation the product and the division by φ are implemented by simple algorithms.

Notation. Given a sequence of binary digits $\alpha = \alpha_1 : \alpha_2 : \ldots$ we indicate with $[\![\alpha]\!]_R$ the real number $\sum_{i \in I\!N^+} \alpha_i \times \varphi^{-i}$.

8.4.1 Addition

The algorithm executes the computation from left to right (from the most meaningful digits to the least meaningful ones) and uses a carry consisting of two digits. At each step, the algorithm needs to examine at most two digits of each addend in order to determine which step to execute.

Formally we define a function A having the following behavior:

1) The inputs of A are two infinite binary sequences α and β, and a string of two binary digits $x : y$ (the carry).

2) The following equality holds

$$[\![A(\alpha,\, \beta,\, x : y)]\!]_R = ([\![\alpha]\!]_R + [\![\beta]\!]_R + [\![x : y]\!]_R)/\varphi^2$$

A is defined by a set of stream recursion equations. From this set of equations we can derive immediately a set of rewriting rules describing the algorithm.

To simplify the description and to stress the important points we write the equations up to permutations between digits of the arguments having equal weight. We use also the symbols x and y as metavariables ranging over binary digits.

With this convention the equation:
$A(0 : x : \alpha,\, 1 : y : \beta,\, 1 : 1) = 1 : A(x : \alpha,\, y : \beta,\, 0 : 0)$
subsume the 8 equations:
$A(1 : x : \alpha,\, 0 : y : \beta,\, 1 : 1) = 1 : A(x : \alpha,\, y : \beta,\, 0 : 0)$
$A(1 : x : \alpha,\, 1 : y : \beta,\, 0 : 1) = 1 : A(x : \alpha,\, y : \beta,\, 0 : 0)$
$A(0 : 1 : \alpha,\, 1 : y : \beta,\, 1 : x) = 1 : A(1 : \alpha,\, y : \beta,\, 0 : x)$
$A(1 : 1 : \alpha,\, 0 : y : \beta,\, 1 : x) = 1 : A(1 : \alpha,\, y : \beta,\, 0 : x)$
$A(1 : 1 : \alpha,\, 1 : y : \beta,\, 0 : x) = 1 : A(1 : \alpha,\, y : \beta,\, 0 : x)$
$A(0 : x : \alpha,\, 1 : 1 : \beta,\, 1 : y) = 1 : A(x : \alpha,\, 1 : \beta,\, 0 : y)$
$A(1 : x : \alpha,\, 0 : 1 : \beta,\, 1 : y) = 1 : A(x : \alpha,\, 1 : \beta,\, 0 : y)$
$A(1 : x : \alpha,\, 1 : 1 : \beta,\, 0 : y) = 1 : A(x : \alpha,\, 1 : \beta,\, 0 : y)$

there equations are obtained by permuting the first digits of the three arguments $\{0, 1, 1\}$ and, *separately*, the second digits $\{x, y, 1\}$

Definition 22 *The function A which implements addition between binary sequences is defined by the following set of equations:*
1) $A(0 : \alpha,\, 0 : \beta,\, 0 : x) = 0 : A(\alpha,\, \beta,\, x : 0)$
2) $A(1 : x : \alpha,\, 0 : y : \beta,\, 0 : 0) = 0 : A(x : \alpha,\, y : \beta,\, 1 : 1)$
3) $A(1 : 1 : \alpha,\, 0 : 1 : \beta,\, 0 : 1) = 1 : 0 : A(\alpha,\, \beta,\, 0 : 1)$
4) $A(0 : 0 : \alpha,\, 1 : 0 : \beta,\, 1 : 0) = 0 : 1 : A(\alpha,\, \beta,\, 1 : 0)$
5) $A(0 : x : \alpha,\, 1 : y : \beta,\, 1 : 1) = 1 : A(x : \alpha,\, y : \beta,\, 0 : 0)$
6) $A(1 : \alpha,\, 1 : \beta,\, 1 : x) = 1 : A(\alpha,\, \beta, x : 1)$

Note. It is interesting to observe than in the set of equations there is a duality between 0 and 1. If, in equations 1), 2) and 3), we substitute the digit 0 with 1 and vice versa we obtain respectively the equations 6), 5) and 4).

Proposition 43 *1) For every α, β, x, y there is one and only one sequence γ such that: $A(\alpha,\, \beta,\, x : y) = \gamma$.*
This amounts to say that A is a well defined function.

2) A has the intended behavior, that is:

$$[\![A(\alpha,\, \beta,\, x : y)]\!]_R = ([\![\alpha]\!]_R + [\![\beta]\!]_R + [\![x : y]\!]_R)/\varphi^2$$

Proof.
1) Easy.

2) We prove by induction on the natural number n that for sequences α, β and digits x, y we have:

$$| \, [\![A(\alpha, \, \beta, \, x : y)]\!]_R - ([\![\alpha]\!]_R + [\![\beta]\!]_R + [\![x : y]\!]_R)/\varphi^2 \, | \leq \varphi^{1-n}$$

The base step of the induction can be proved by a simple computation.

The proof of the induction step is done by cases. In each case we consider a possible assignment to the first two digits of α and β and to x and y.

Alternatively we can prove this by showing that the induction step is valid for all cases subsumed by each one of the 6 equations.

Here we present only the proof for equation 5). The proofs for the other equations follow a similar pattern.

The following equalities are trivially true:

$$([\![0 : x : \alpha]\!]_R + [\![1 : y : \beta]\!]_R + [\![1 : 1[\![_R)/\varphi^2 =$$

$$= ([\![x : \alpha]\!]_R/\varphi + [\![y : \beta]\!]_R/\varphi + 1/\varphi + 1/\varphi + 1/\varphi^2)/\varphi^2 =$$

$$= ([\![x : \alpha]\!]_R/\varphi + [\![y : \beta]\!]_R/\varphi + 1/\varphi + 1)/\varphi^2 =$$

$$= ([\![x : \alpha]\!]_R/\varphi + [\![y : \beta]\!]_R/\varphi + \varphi)/\varphi^2 =$$

$$= 1/\varphi + ([\![x : \alpha]\!]_R/\varphi + [\![y : \beta]\!]_R/\varphi)/\varphi^2$$

and

$$[\![A(0 : x : \alpha, \, 1 : y : \beta, \, 1 : 1)]\!]_R =$$

$$= [\![1 : A(x : \alpha, \, y : \beta, \, 0 : 0)]\!]_R =$$

$$= 1/\varphi + [\![A(x : \alpha, \, y : \beta, \, 0 : 0)]\!]_R/\varphi =$$

finally by the above equalities and by induction hypothesis we get:

$$| \, [\![A(0 : x : \alpha, \, 1 : y : \beta, \, 1 : 1)]\!]_R - ([\![0 : x : \alpha]\!]_R + [\![0 : y : \beta]\!]_R + [\![x : y]\!]_R)/\varphi^2 \, | =$$

$$(| \, [\![A(x : \alpha, \, y : \beta, \, 0 : 0)]\!]_R - ([\![x : \alpha]\!]_R + [\![y : \beta]\!]_R + [\![0 : 0]\!]_R)/\varphi^2)/\varphi \, | \leq$$

$$\varphi^{1-(n-1)}/\varphi = \varphi^{1-n}$$

□

8.4.2 Subtraction.

We define a function "complement" C:

Definition 23 *The function C is defined by the equations:*
1) $C(1 : \alpha) = 0 : C(\alpha)$
2) $C(0 : \alpha) = 1 : C(\alpha)$

Proposition 44 *1) C is a well defined function.*
2) For every sequence α

$$[\![C(\alpha)]\!]_R = \varphi - [\![\alpha]\!]_R$$

Proof.
1) Trivial.
2) Follows from the equalities:

$$[\![\alpha]\!]_R + [\![C(\alpha)]\!]_R = \sum_{i \in N^+} 1/\varphi^i = 1/(1-\varphi) = \varphi$$

□

The above property of the function C can then be used to easily obtain an algorithm for subtraction.

8.4.3 Multiplication.

An easy way to obtain an algorithm for multiplication is to reduce the multiplication to a series of additions.

Definition 24 *The function P between sequences of digits is defined by the following set of equations:*
1) $P(0 : \alpha,\ \beta) = 0 : P(\alpha,\ \beta))$
2) $P(\alpha,\ 0 : \beta) = 0 : P(\alpha,\ \beta))$
3) $P(1 : 0 : \alpha,\ 1 : 0 : \beta) = 0 : A(A(\alpha,\ \beta,\ 0 : 0),\ 0 : P(\alpha,\ \beta),\ 1 : 0)$
4) $P(1 : 1 : \alpha,\ 1 : 0 : \beta) = A(A(0 : \alpha,\ \beta,\ 0 : 0),\ 0 : 0 : P(\alpha,\ \beta),\ 1 : 0)$
5) $P(1 : 0 : \alpha,\ 1 : 1 : \beta) = A(A(\alpha,\ 0 : \beta,\ 0 : 0),\ 0 : 0 : P(\alpha,\ \beta),\ 1 : 0)$
6) $P(1 : 1 : \alpha,\ 1 : 1 : \beta) = A(A(\alpha,\ \beta,\ 0 : 0),\ 0 : 0 : P(\alpha,\ \beta),\ 1 : 1)$

Proposition 45 *1) P is a well defined function.*
2) For every sequences α and β the following equality holds:

$$[\![P(\alpha,\ \beta)]\!]_R = \frac{[\![\alpha]\!]_R \times [\![\beta]\!]_R}{\varphi^2}$$

Proof

1) Easy.

2) We argue as for the case of addition. By induction on the natural number n it is possible to prove that for every natural number n and for every pair of sequences α and β the following equality holds:

$$\mid [\![P(\alpha,\ \beta)]\!]_R - \frac{[\![\alpha]\!]_R \times [\![\beta]\!]_R}{\varphi^2} \mid \leq\ \varphi^{1-n}$$

□

The time necessary to evaluate the first n digits of the multiplication using the algorithm described above is proportional to n^2. On the other hand the time complexity is linear for the algorithm implementing addition.

It is possible to define more elaborated algorithms for multiplication which are based on the fast Fourier transform, whose order of complexity is $n \times \log n \times \log\log n$ [19]. But we do not discuss them here for lack of time.

8.4.4 Division.

An algorithm for the division, valid for the full notation, can be obtained from the function D described here.

Definition 25 *The function D between sequences of digits is defined by the following equations:*

0) $D(\alpha,\ 1:\beta) = D_1(0:0:\alpha,\ C(\beta))$

1) $D_1(0:0:\alpha,\ \beta) = D_2(A(\alpha,\ 1:\beta,\ 1:1),\ 0:\alpha,\ \beta)$

2) $D_1(0:1:\alpha,\ \beta) = D_2(A(\alpha,\ 0:\beta,\ 1:1),\ 1:\alpha,\ \beta)$

3) $D_1(1:0:\alpha,\ \beta) = D_2(A(\alpha,\ 1:\beta,\ 1:1),\ \alpha,\ \beta)$

4) $D_2(0:0:\gamma,\ \alpha,\ \beta) = 0 : D_1(\alpha, \beta)$

5) $D_2(0:1:0:\gamma,\ \alpha,\ \beta) = 0 : D_1(\alpha,\ \beta)$

6) $D_2(0:1:1:\gamma,\ \alpha,\ \beta) = 1 : D_1(0:0:\gamma,\ \beta)$

7) $D_2(1:\gamma,\ \alpha,\ \beta) = 1 : D_1(\gamma,\ \beta)$

Proposition 46 *For every pair of sequences α and β such that β has an initial subsequence of the form $1:1$ or $1:0:1$ we have:*

1) there is only a sequence γ such that: $D(\alpha,\ \beta) = \gamma$;

2) the following equality holds:

$$[\![D(\alpha,\ \beta)]\!] = \frac{[\![\alpha]\!]}{[\![\beta]\!] \times \varphi^3}$$

There is a restriction in the use of the function D. D evaluates the division between α and β only if β begins with $1,1$ or $1,0,1$. This is not a severe restriction. In fact, given a string β, it is easy to obtain a string β' and a

natural number n such that $[\![\beta]\!]_R \times \varphi^n = [\![\beta']\!]_R$ and where β' begins with $1,1$ or $1,0,1$.

The algorithm for division is defined by few equations, but its correctness is quite difficult to prove. To reduce the number of equations employed many arithmetic properties have to be used.

The algorithm is based on the euclidean algorithm for division. Roughly speaking, the idea is the following: to obtain the result of the division of α by β we consider first the value $[\![\alpha]\!] \times \varphi - [\![\beta]\!]$ and then we try to determine if this number is larger than zero or if it smaller than $[\![0101]\!]_R$. In the first case we can safely generate 1 as first digit followed by the result of the division of $([\![\alpha]\!] \times \varphi - [\![\beta]\!]) \times \varphi$ by β. In the second case we can safely generate as first digit 0 followed by the result of the division of $([\![\alpha]\!]\times)\varphi$ by β.

8.5 A "Large digit" representation

In this section we present an elaborate generalization of the negative digit notation for real numbers.

The motivation for defining a such notation is to improve significantly the efficiency of the real number computation when lazy functional languages are used.

The main idea is the following. In almost every computer the arithmetic operation on the integer are implemented quite efficiently. We want therefore to take as much advantage as possible from this fact.

As a first attempt to obtain this result we try use a negative digit notation having a "large" base. Let us compare two negative digit notations; one having as base a number smaller than ten, and the other having as base a number with the same magnitude of the biggest integer representable by a word in the computer. In the large base notation fewer digits are necessary to determine the value of a real number with a given precision. The time necessary to evaluate a new digit in the result of an arithmetic operation does not change significantly between the two notations. This is so because the computation of a new digit is mainly a sequence of arithmetic operation on integers. So fewer steps are necessary to obtain the result of arithmetic operations, with a given precision, when the large base is used.

This approach is analyzed in [6] where an significant difficulty is pointed out. The best way to explain it is by an example. In every negative digit notation in order to determine n digits (after the dot) in a sum $(a + b)$ it is necessary to determine $n + 1$ digits of a and $n + 1$ digit of b. Let us consider a program P that evaluates the sum $(a_1 + (a_2 + (\ldots + a_i)\ldots)$ performing $i - 1$ additions in the order defined by the parentheses. Observe that P needs to examine i digits of the value a_i to determine the first digit of the sum.

If the value of the base is large and if the number of addends is significant, then in order to determine the first digit of the sum the program P needs to determine the value of a_i with a great precision. But such a great precision is

not strictly necessary. If the value a_i is in turn the result of a procedure then not strictly necessary extra computation has to be executed.

Phenomena of this kind are likely to happen in many other contexts. This fact cancels all advantages deriving from a the large base notation. Using the words of [6] the large base notations has a too large "granularity". In [6] it is suggested a possible solution for overcoming this difficulty connected to the large granularity, which consists of a modification the lazy evaluation mechanism.

Here we present a solution of the "granularity problem" that goes in a different direction. We intend to maintain the standard lazy evaluation and modify instead the real number notation.

Definition 26 *For every natural number $b > 1$ the* digit-error notation *with base b is defined as follows: a real number r is represented by a sequence of pairs of integers $[d_0, e_0], [d_1, e_1], \ldots$ such that:*
i) $\forall i \in N^+. \; -b^2 < d_i < b^2$
ii) $\forall i \in N. \; -b \leq e_i \leq b$
iii) $\forall i \in I\!N. e_i \geq \sum_{j \in I\!N^+} d_{i+j} \times b^{-j}$
iv) $r = \sum_{i \in I\!N} d_i \times b^{-i}$

In the digit-error notation an infinite sequence, having as finite subsequence $[d_0, e_0], \ldots, [d_n, e_n]$ denotes a real number bigger than
$(\sum_{i=0}^{n} d_i \times b^{-i}) - e_n \times b^{-n}$
and smaller than
$(\sum_{i=0}^{n} d_i \times b^{-i}) + e_n \times b^{-n}$.

The idea underlying this definition of the digit-error is simple: the approximation with which the first $n+1$ digits determine the value may change. The values e_n indicate this precision. In other words the e_n's give a limit to the error contained in the digit d_n. The $n+1$-th digit corrects the errors contained in the n-th digit.

It is not difficult to prove that it is possible to go in a effective way from a digit-error notation to the negative digit notation. As a consequence, from the theoretical point of view, the error-digit notation is therefore completely equivalent to the other forms of real notation in definition 5.

Going back to the previous example we can see now how the problem of the granularity is solved. In order to determine the n-th digit of $a+b$ it is not always necessary to know the first $n+1$ digits of a and b. If the error values associated to the n-th digits of a and b are sufficiently small then it is not necessary to consider the $n+1$-th digits of a and b, the n-th digits are sufficient.

Using this digit-error notation, the algorithms are not any more forced to evaluate the arguments with a precision that is much higher than the strictly necessary.

As example, we present in appendix A a set of algorithms implementing arithmetic operations in the digit-error notations.

These algorithms can be considered a modification of the standard algorithms for the arithmetic operations. They are inspired by the same ideas.

The chosen programming language is "Scheme". We employ an implementation of "Scheme" supporting arbitrary-large integer number computation. This capability has been used in writing the programs.

In writing the programs we privilege simplicity in place of efficiency. More elaborated programs can lead to more efficient implementations of the arithmetic functions.

For the moment we did not test the efficiency of the algorithms.

Chapter 9

Conclusions and directions for further works

In this work we analyzed the exact real number computations in functional programming languages.

The thesis shows as domain theory can be usefully employed to carry on an analysis of computability on real numbers. A limit of domain theory has been pointed out: using Scott-domain we cannot obtained a completely faithful representation of the real line. The main result presented in the thesis is a topological characterization of the computable functionals on real numbers: we show that every computable functional is continuous w.r.t. the compact open topology on the functions space.

A second important result in the thesis concerns the issue of realizing an abstract data type for the real numbers: we shown that such a data type exists also if it can not be obtained using only sequential primitives.

In the thesis we presented moreover new alternative representation for the real numbers. This representations can be usefully employed in some approaches to the effective implementation of the real number computations. Particularly interesting is the representation using as base the golden ratio.

Further direction of research are suggested by this work: in the thesis there are several open problems that need to be investigated. Particularly interesting is the question whether sequential computation is sufficiently powerful to generate all the computable functionals on real number.

A second direction for further research is the to consider implementations of the real number computation in other setting different from the classical functional programming languages. Particularly appealing is to considered the real number computation realized in a second order lambda calculus or, in a logical framework. In these cases we can have languages where all the computations converge and sufficiently rich to express all the useful functions on real numbers. It is an interesting issue to characterize the functions that are computable in these settings. Moreover the problem of the existence of an abstract data

type for the real number can be reconsidered for these cases. In particular one should investigate whether the sequential computation is sufficient to realize an abstract data type in these settings.

Finaly we intend to develop further the implementation of the real number computation. In particular we intend to investigate whether the large base representation can be a feasible approach to the practical real number computation. To do that it is necessary to improve the efficiency of the algorithms for the arithmetic functions presented in the thesis and to develop the algorithms for the all the basic the analytic functions.

Acknowledgements

A significant part of this thesis was originally done under the supervision of Michael B. Smyth. I thank him for many interesting and illuminating discussions. I thank also the other members of the "real group" at Imperial College in London, Wilson De Olivera and Ian Stewart.

I would like to acknowledge also Roberto Amadio, Piero D'Ancona, Simone Martini, Per Martin-Löf, Steve Vickers and most of all Giuseppe Longo for the suggestions that they gave over the years. The comments and suggestions from referees Robert Cartwright and Thierry Coquand helped me in improving the first version of the thesis.

Finally I thank my advisor Furio Honsell for the encouragement and for directing this work.

Bibliography

[1] O. Aberth, "Computable Analysis." MacGraw-Hill, New-York 1980.

[2] A. Avizienis, "Binary-Computable Signed-Digit Arithmetic." AFIPS Conference Proceedings 26,1 (1964) 663-672.

[3] L.E.J. Brouwer, "Beweis, dass jede volle Funktion gleichmässig stetig ist" Proc. Amsterdam 27 (1924) 189-194.

[4] M.J. Beeson, "Foundation of Constructive Mathematics" Spriger-Verlag, Berlin, 1985.

[5] E. Bishop, "Foundation of Constructive Analysis." McGraw-Hill, New York, 1967.

[6] H.-J. Boehm, R. Cartwright, M. Riggle, and M.J. O'Donell, "Exact Real Arithmetic: A Case Study in Higher Order Programming." 1986 ACM Symposium on Lisp and Functional Programming.

[7] H.-J. Boehm, "Constructive Real Interpretation of Numerical Programs." SIGPLAN Notice 22, 7 (July 87), pp. 214-221.

[8] H.-J. Boehm, R. Cartwright, "Exact Real Arithmetic: Formulating Real Numbers as Functions" in "Research Topics in Functional Programming" David Turner editor, Addison-Wesley, 1990, pp. 43-64

[9] D. Bridges and E. Bishop, "Constructive Analysis." Springer-Verlag, Berlin, 1985.

[10] K.E. Grue, "Unrestricted Lazy Numerical Algorithms." Unpublished paper.

[11] A. Grzegorczyk, "On the Definition of Computable Real Continuous Functions." Fund. Math. 44 (1957) 61-77.

[12] K. Ko and Friedmann, "Computational Complexity of Real Functions." Theoret. Comput. Sci. 20 (1982) 323-352.

[13] K. Ko, "Reducibilities on Real Numbers." Theoret. Comput. Sci. (1984).

[14] C. Kreitz, K. Weihrauch, "Theory of representation" Theoret. Comp. Sci. 38 (1985) 35-53.

[15] D. Lacombe, "Quelques procédés de définitions en topologie recursif." in: Constructivity in Mathematics, North-Holland (1959) 129-158.

[16] P. Martin-Löf, "Note on Constructive Mathematics." Almqvist and Wiksell, Stockholm (1970).

[17] J. Myhill, "Criteria of Constructibility for Real Numbers." J. Symbolic Logic 18 (1953) 7-10.

[18] J. Myhill, "What is a Real Number?" America Mathematical Monthly (1979) 748-754.

[19] N. Th. Müller, "Subpolinomial Complexity Classes of Real Functions and Real Numbers." Lectures Notes in Computes Science 226, Springer-Verlag, Berlin (1987).

[20] M.B. Pour-El and J.I. Richards "Computability in Anality in Analysis and Phisics." Springer-Verlag, New York (1990).

[21] G.D. Plotkin, "LCF Considered as a Programing Language." Theoret. Comput. Sci. 5 (1977) 223-255.

[22] H.G. Rice, "Recursive Real Numbers." Proc. Amer. Math. Soc 5 (1954) 784-791.

[23] H. Rogers, Jr., "Theory of recursive function and effective computability." MacGraw-Hill, New York (1967).

[24] W. Rudin, "Principles of Mathematical Analysis" MacGraw-Hill, New York (1964).

[25] D. Scott, "Lattice Theory, Data Types and Semantics."

[26] D. Scott, "Outline of the Mathematical Theory of Computation." Proc. 4th Princeton Conference on Information Science (1970).

[27] D. Scott, "Data Types as Lattices." SIAM J. Comput. 5 (1976) 522-587.

[28] M.B. Smyth, "Effectively given Domains." Theoret. Comp. Sci 2 (1976) 257-274.

[29] M.B. Smyth, "Quasi-uniformities: Reconciling Domains and Metric Spaces." (Tulane, 1987), LNCS 1988.

[30] E. Specker, "Nicht konstruktiv beweisbare Satze der Analysis." J. of Symbolic Logic 14 (1949) 145-158.

[31] A.S. Troelstra and D. van Dalen, "Constructivism in Mathematics." North-Holland, Amsterdam (1988).

[32] A.M. Turing, "On Computable Numbers, with an Application to the Entscheidungs Problem." Proc. London Math. Soc. 42 (1937) 230-265.

[33] J. Vuillemin, "Exact Real Computer Arithmetic with Continued Fraction." Proc. A.C.M. conference on Lisp and functional Programming (1988) 14-27.

[34] K. Weihrauch, U. Schreiber, "Embeding Metric Spaces into cpo's" Theoret. Comp. Sci. 16 (1981) 5-34.

[35] K. Weihrauch and C. Kreitz, "Representation of the Real Numbers and of the Open Subsets of the Set of Real Numbers." Annals of Pure and Applied Logic 35 (1987) 247-260.

[36] E. Wiedmer, "Computing with Infinite Objects." Theoret. Comp. Sci. 10 (1980) 133-155.