Models of computability of partial functions on the reals

By
Ming Quan Fu, B.Sc.

A Thesis
Submitted to the School of Graduate Studies
in partial fulfilment of the requirements for the degree of

Master of Science
Department of Computing and Software
McMaster University

MASTER OF SCIENCE (2007) McMaster University
(Computing and Software) Hamilton, Ontario

TITLE: Models of computability of partial functions on the reals

AUTHOR: Ming Quan Fu, B.Sc. (Daqing Petroleum Institute, PRC)

SUPERVISOR: Dr. Jeffery I. Zucker

NUMBER OF PAGES: vii, 68

Abstract

Various models of computability of partial functions f on the real numbers are studied: two abstract, based on approximable computation w.r.t high level programming languages; two concrete, based on computable tracking functions on the rationals; and two based on polynomial approximation. It is shown that these six models are equivalent, under the assumptions: (1) the domain of f is a union of an effective sequence of rational open intervals, and (2) f is effectively locally uniformly continuous. This includes the well-known functions of elementary real analysis (rational, exponential, trigonometric, etc., and their inverses) and generalises a previously know equivalence result for total functions on the reals.

Acknowledgements

I would first like to express my sincere thanks and appreciation from the bottom of my heart to my supervisor, Dr. J.I. Zucker, for his insight, thoughtful guidance and constant encouragement throughout my study.

I am grateful to the members of my Examination Committee, Dr. E. Sekerinski and Dr. J. Carette, for their careful review and valuable comments.

Thanks to my friends in ITB 206 for their companionship during my graduate studies.

Last, but not least, thanks to my mother, my late father, and my sisters, for all their support and encouragement over the years.

Contents

Chapter 1

Introduction

1.1 Overview of the thesis

In this thesis, various models of computability of partial functions on the real numbers are studied: two abstract, based on approximable computation w.r.t high level programming languages; two concrete, based on computable tracking functions on the rationals; and two based on polynomial approximation.

In [TZ05] it was shown that a number of such models are equivalent for total functions on $\mathbb{R}$, under the assumption of effective local uniform continuity. It was conjectured there that the result also holds for partial function. In this thesis, we prove that conjecture, for partial functions on $\mathbb{R}$ whose domain is a union of an effective sequence of rational open intervals, which are effectively locally uniformly continuous.

We shall prove, under these conditions, the equivalence of six models:

(*i*) ***While***$^*(R_t^N)$ approximable computability (where R_t^N is a total algebra on $\mathbb{R}$);

(*ii*) ***WhileCC***$^*(R_p^N)$ approximable computability (where R_p^N is a partial algebra on $\mathbb{R}$);

(*iii*) GL-(Grzegorczyk/Lacombe) computability;

(*iv*) $\bar{\alpha_0}$-computability (where α_0 is a standard enumeration of the rationals);

(*v*) Effective local uniform $\mathbb{Q}$-polynomial approximability;

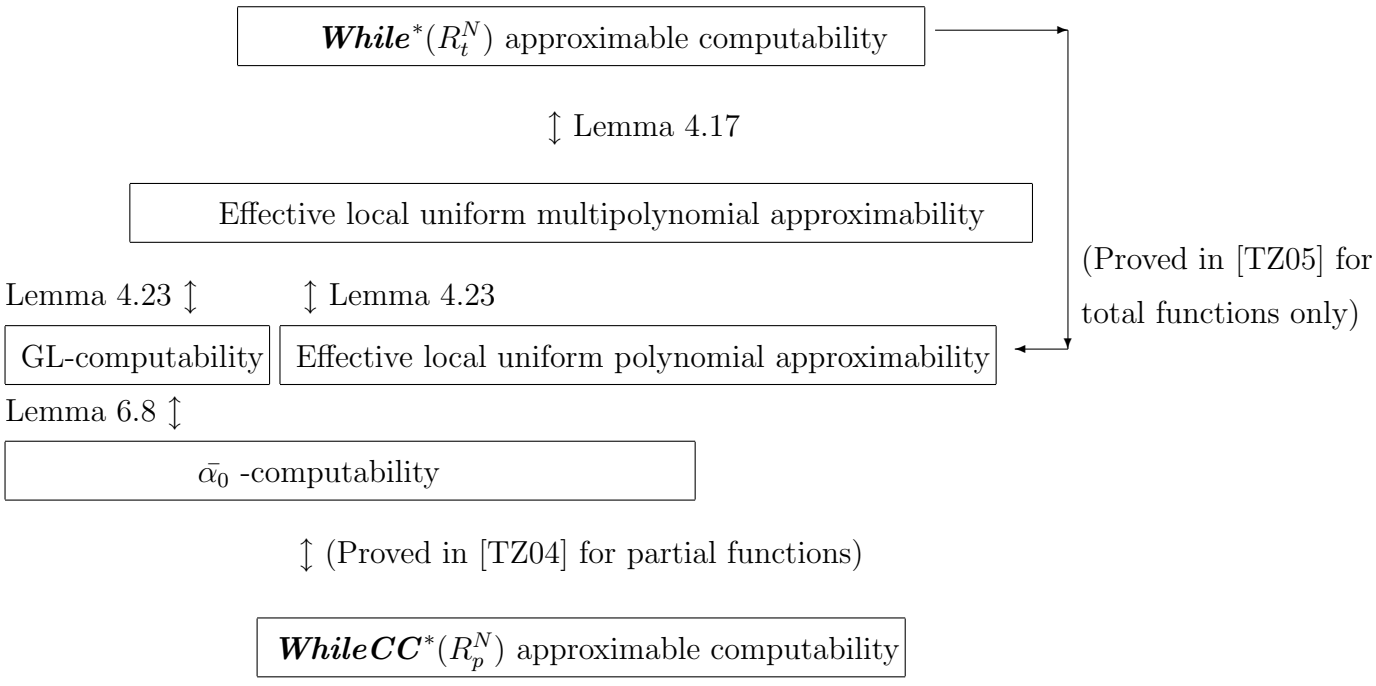
(*vi*) Effective local uniform $\mathbb{Q}$-multipolynomial approximability.

Models (*i*) and (*ii*) are abstract and (*iii*) and (*iv*) are concrete.

Model (*vi*), multipolynomial approximation, is a new model, not considered in [TZ05], which enables us to generalize the equivalence result to partial functions.

The last two named procedures are illustrated using Maple 9.5.

We will prove the equivalence theorem for partial functions as in the following diagram:



In Chapter 2, we give some basic concepts we will use later, such as N-standard many-sorted signatures, algebras, topological, and metric partial algebras.

In Chapter 3, we give the syntax and semantics of the $\boldsymbol{While}^{(*)}$ programming languages[1] as defined in [TZ04]. These form the basic of the first of our two abstract computation models on $\mathbb{R}$: $\boldsymbol{While}^{(*)}$ approximable computability on the total metric algebra R_t^N, and we also give the definition of exhaustion and $\boldsymbol{While}^{(*)}$ approximability.

In Chapter 4, we define the concepts of $\mathbb{Q}$-polynomial and $\mathbb{Q}$-multipolynomial approximability, and also give the first concrete computation model, GL-computability. We also prove the equivalence of the following models of computation for partial functions:

(i) $\boldsymbol{While}^*(R_t^N)$ approximable computability;

(ii) GL-(Grzegorczyk/Lacombe) computability;

(iii) Effective local uniform $\mathbb{Q}$-polynomial approximability;

(vi) Effective local uniform $\mathbb{Q}$-multipolynomial approximability.

We also give examples of polynomial and multipolynomial approximability for some well-known partial functions.

In Chapter 5, we explain the syntax and semantics of the ***WhileCC*** programming language (i.e. ***While*** with "countable choice"), which forms the basic of the second of our two abstract computation models on $\mathbb{R}$, using the partial metric algebra $\mathbb{R}_p^N$.

In Chapter 6, we present our second concrete model, $\bar{\alpha_0}$-computability, and we give the theorem proving the equivalence of all six models of computability on $\mathbb{R}$ under the assumptions stated above.

---

[1] $\boldsymbol{While}^*$ means ***While*** with arrays. $\boldsymbol{While}^{(*)}$ means ***While*** with or without arrays.

It should be noted that these assumptions on a function f (namely (1) the domain of f is a union of an effective sequence of rational open intervals, and (2) f is effectively locally uniformly continuous w.r.t. this sequence) hold for all the well-known functions of elementary real analysis (rational, exponential, trigonometric, etc., and their inverses) including, for example, $f(x) = \sin \frac{1}{x}$ with domain $U = \{x \in \mathbb{R} \mid x \neq 0\}$.

The most challenging part of this thesis turned out to be the proof of Lemma 4.23 and the preparations leading up to it in Section 4.2. It is here that we were able to generalize the results of [TZ05] to partial functions.

1.2 Abstract and concrete computability

In the theory of computation on topological algebras, there is a considerable gap between so-called abstract and concrete models of computation. Abstract models of computation are independent of data representations, while concrete models depend on data representations [TZ05].

In this thesis, we use the abstract computation models defined in [TZ00, TZ04], based on a high level imperative programming language ***While***. Many abstract models of computation have been defined and shown to be equivalent over general algebras. So, this theory of abstract computation is stable. For a comprehensive introduction to abstract computation, see [TZ00, TZ04].

The relationship between various concrete models are fairly well understood [TZ05]. For example, a number of concrete models including TTE ("Type Two Enumeratility") of [Wei00] has been shown to be equivalent to GL-Computability [SHT99]. On the other hand the inequivalences between three well-known models: Markov, Banach-Mazur and GL-computability have recently been proved in

[Her05, Her06].

However, the connection between abstract and concrete theories has long been problematic. Recently, the situation has been clarified [TZ04]. For example, the equivalence between the abstract model $\boldsymbol{WhileCC}(R_p^N)$ approximability and the concrete model $\bar{\alpha_0}$ computability for partial function was proved in [TZ04]; we merely quote the result in this thesis.

Chapter 2

Signatures and Topological Partial Algebras

In this Chapter, we give some basic concepts which will be used in this thesis, such as signature, partial algebras and topological metric algebra. Most of the definitions are from [TZ04].

2.1 Signatures

Definition 2.1 (Many-sorted signatures). A *many-sorted signature* Σ is a pair $\langle \boldsymbol{Sort}(\Sigma), \boldsymbol{Func}(\Sigma) \rangle$ where

(a) $\boldsymbol{Sort}(\Sigma)$ is a finite set of *sorts*,

(b) $\boldsymbol{Func}(\Sigma)$ is a finite set of *(primitive or basic) function symbols* F with

$$\mathsf{F} : \ s_1 \times \cdots \times s_m \to s \qquad (m \geq 0).$$

Each symbol F has a *type* $s_1 \times \cdots \times s_m \to s$, where $m \geq 0$ is the *arity* of F, and $s_1, \ldots, s_m \in \boldsymbol{Sort}(\Sigma)$ are the *domain sorts* and $s \in \boldsymbol{Sort}(\Sigma)$ is the *range sort*

of F. The case $m = 0$ corresponds to *constant symbols*, we then write $\mathsf{F} : \rightarrow s$.

Definition 2.2 (Product types over Σ). A *product type* over Σ, or *Σ-product type*, is a symbol of the form $s_1 \times \cdots \times s_m$ $(m \geq 0)$, where $s_1, \ldots, s_m$ are sorts of Σ, called its *component sorts*.

For a Σ-product type u and Σ-sort s, let $\boldsymbol{Func}\,(\Sigma)_{u \rightarrow s}$ denote the set of all Σ-function symbols of type $u \rightarrow s$.

Definition 2.3 (Function types). Let A be a Σ-algebra. A *function type* over Σ, or *Σ-function type*, is a symbol of the form $u \rightarrow s$, with *domain type* u and *range type* s, where u is a Σ-product type.

Definition 2.4 (Σ-algebras). A *Σ-algebra* A has, for each sort s of Σ, a non-empty set A_s, called the *carrier of sort* s, and for each Σ-function symbol $\mathsf{F} : s_1 \times \cdots \times s_m \rightarrow s$, a (*partial*) function $\mathsf{F}^A : A_{s_1} \times \cdots \times A_{s_m} \rightarrow A_s$.

For a Σ-product type $u = s_1 \times \cdots \times s_m$, we define

$$A^u \;=_{df}\; A_{s_1} \times \cdots \times A_{s_m}.$$

Thus $x \in A^u$ iff $x = (x_1, \ldots, x_m)$, where $x_i \in A_{s_i}$ for $i = 1, \ldots, m$. So each Σ-function symbol $\mathsf{F} : u \rightarrow s$ has an interpretation $\mathsf{F}^A : A^u \rightarrow A_s$. If u is empty, *i.e.*, F is a constant symbol, then F^A is an element of A_s.

The algebra A is *total* if F^A is total for each Σ-function symbol F. Without such a totality assumption, A is called partial. In this thesis we deal mainly with partial algebras.

We will write $\Sigma(A)$ to denote the signature of an algebra A.

Example 2.5 (Booleans). The signature of booleans is important. It can be defined as

signature	$\Sigma(\mathcal{B})$
sorts	bool
functions	$\mathsf{true}, \mathsf{false} : \ \rightarrow \mathsf{bool},$
	$\mathsf{and}, \mathsf{or} : \ \mathsf{bool}^2 \rightarrow \mathsf{bool},$
	$\mathsf{not} : \ \mathsf{bool} \rightarrow \mathsf{bool}$

The algebra $\mathcal{B}$ of booleans contains the carrier $\mathbb{B} = \{\mathsf{tt}, \mathsf{ff}\}$ of sort bool. The algebra $\mathcal{B}$ can therefore displayed as follows:

algebra	$\mathcal{B}$
carriers	$\mathbb{B}$
functions	$\mathsf{tt}, \mathsf{ff} \ \rightarrow \mathbb{B},$
	$\mathsf{and}^{\mathcal{B}}, \mathsf{or}^{\mathcal{B}} : \ \mathbb{B}^2 \rightarrow \mathbb{B},$
	$\mathsf{not}^{\mathcal{B}} : \ \mathbb{B} \rightarrow \mathbb{B},$

Example 2.6 (Naturals). The signature of naturals is defined as

signature	$\Sigma(\mathcal{N}_0)$
sorts	nat
functions	$0 : \ \rightarrow \mathsf{nat}$,
	$\mathsf{suc} : \ \mathsf{nat} \rightarrow \mathsf{nat}$

The algebra $\mathcal{N}_0$ of naturals has a carrier $\mathbb{N}$ of sort nat, together with the constant zero and successor function:

algebra	$\mathcal{N}_0$
sorts	$\mathbb{N}$
functions	$0^{\mathsf{N}} : \ \rightarrow \mathbb{N}$,
	$\mathsf{suc}^{\mathsf{N}} : \ \mathbb{N} \rightarrow \mathbb{N}$

Example 2.7 (Reals). The ring $\mathcal{R}_0$ of reals has a carrier $\mathbb{R}$ of sort real:

algebra	$\mathcal{R}_0$
carriers	$\mathbb{R}$
functions	$0, 1 : \ \rightarrow \mathbb{R}$,
	$+, \times : \ \mathbb{R}^2 \rightarrow \mathbb{R}$
	$- : \mathbb{R} \rightarrow \mathbb{R}$

The field $\mathcal{R}_1$ of reals has a carrier $\mathbb{R}$ of sort real:

algebra	$\mathcal{R}_1$
import	$\mathcal{R}_0$
functions	$\mathsf{inv}^{\mathcal{R}} : \mathbb{R} \to \mathbb{R},$

where

$$\mathsf{inv}^{\mathcal{R}}(x) \quad = \quad \begin{cases} 1/x & \text{if } x \neq 0 \\ \uparrow & \text{otherwise.} \end{cases}$$

The algebra $\mathcal{R}_1$ is thus a partial algebra.

Definition 2.8 (Reducts and expansions). Let Σ and Σ' be signatures.

(a) We write $\Sigma \subseteq \Sigma'$ to mean $\boldsymbol{Sort}(\Sigma) \subseteq \boldsymbol{Sort}(\Sigma')$ and $\boldsymbol{Func}\,(\Sigma) \subseteq \boldsymbol{Func}(\Sigma')$.

(b) Suppose $\Sigma \subseteq \Sigma'$. Let A and A' be algebras with signatures Σ and Σ' respectively.

- The Σ-*reduct* $A'|_\Sigma$ of A' is the algebra of signature Σ, consisting of the carriers of A' named by the sorts of Σ and equipped with the functions of A' named by the function symbols of Σ.
- A' is a Σ'-*expansion* of A if and only if A is the Σ-*reduct* of A'.

Definition 2.9 (Σ-variables). For each Σ-sort s, there are (program) variables a^s, b^s, ..., $\mathtt{x}^s$, $\mathtt{y}^s$, ... of sort s. Let $\boldsymbol{Var}_s(\Sigma)$ be the class of variables of sort s, and $\boldsymbol{Var}(\Sigma)$ be the class of all Σ-variables, $\mathtt{x}$, $\mathtt{y}$,... ...

Definition 2.10 (Σ-terms). Let $\boldsymbol{Term}(\Sigma)$ be the class of Σ-terms $t, \ldots$, and $\boldsymbol{Term}_s$ be the class of terms of sort s, defined by

$$t^s ::= \mathsf{x}^s \mid \mathsf{F}(t_1^{s_1}, \ldots, t_m^{s_m}) \mid \mathsf{if}\ b\ \mathsf{then}\ t_1^s\ \mathsf{else}\ t_2^s\ \mathsf{fi}$$

where $\mathsf{F} \in \boldsymbol{Func}\,(\Sigma)_{u \to s}$ and $u = s_1 \times \cdots \times s_m$. We write t^s or $t : s$ to indicate that $t \in \boldsymbol{Term}_s$. Further, we write $t : u$ to indicate that t is a u-tuple of terms, *i.e.*, a tuple of terms of sorts $s_1, \ldots, s_m$.

We will often write $\boldsymbol{Var}$ for $\boldsymbol{Var}(\Sigma)$, $\boldsymbol{Term}$ for $\boldsymbol{Term}(\Sigma)$, etc.

Definition 2.11 (Closed terms over Σ). We define the class $\boldsymbol{CT}(\Sigma)$ of *closed terms over* Σ, and for each Σ-sort s the class $\boldsymbol{CT}(\Sigma)_s$ of closed terms of sort s. These are generated inductively by the rule: if $\mathsf{F} \in \boldsymbol{Func}\,(\Sigma)_{u \to s}$, and $t_i \in CTSig_s$ for $i = 1, \ldots, m$ where $u = s_1 \times \cdots \times s_m$, then $\mathsf{F}(t_1, \ldots, t_m) \in CTSig_s$.

Note that the implicit base case of this inductive definition is the case where $m = 0$, which yields: for all constants $\mathsf{c} : \to s$, $\mathsf{c}() \in \boldsymbol{CT}(\Sigma)_s$. In this case we write c instead of $\mathsf{c}()$. Hence if Σ contains no constants, $\boldsymbol{CT}(\Sigma)$ is empty.

Assumption 2.12 (Instantiation) In this thesis, we will assume:

$\boldsymbol{CT}(\Sigma)_s$ *is non-empty for each* $s \in \boldsymbol{Sort}(\Sigma)$.

Definition 2.13 (Valuation of closed terms). For a Σ-algebra A and $t \in \boldsymbol{CT}(\Sigma)_s$, we define the *valuation* $t_A \in A_s$ of t in A by structural induction on t:

$$\begin{aligned} \mathsf{F}(t_1,\ldots,t_m)_A &\simeq \mathsf{F}^A((t_1)_A,\ldots,(t_m)_A) \\ (\mathsf{if}\ b\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2\ \mathsf{fi})_A &\simeq \begin{cases} (t_1)_A & \text{if } b_A{\downarrow}\mathtt{tt} \\ (t_2)_A & \text{if } b_A{\downarrow}\mathtt{ff} \\ \uparrow & \text{otherwise.} \end{cases} \end{aligned}$$

where " $\simeq$ " means that either both sides converge to same value or diverge, and "$\uparrow$" means undefined or diverges ("$\downarrow$" means converges).

In particular, for $m = 0$, *i.e.*, for a constant $\mathsf{c} : \to s$, $\mathsf{c}_A = \mathsf{c}^A$.

Definition 2.14 (Default terms; Default values).

(a) For each sort s, we pick a closed term of sort s. (There is at least one, by the Instantiation Assumption.) We call this the *default term of sort* s, written $\boldsymbol{\delta}^s$. Further, for each product type $u = s_1 \times \cdots \times s_m$ of Σ, the *default (term) tuple of type* u, written $\boldsymbol{\delta}^u$, is the tuple of default terms $(\boldsymbol{\delta}^{s_1},\ldots,\boldsymbol{\delta}^{s_m})$.

(b) Given a Σ-algebra A, for any sort s, the *default value of sort* s *in* A is the valuation $\boldsymbol{\delta}^s_A \in A_s$ of the default term $\boldsymbol{\delta}^s$; and for any product type $u = s_1 \times \cdots \times s_m$, the *default (value) tuple of type* u *in* A is the tuple of default values $\boldsymbol{\delta}^u_A = (\boldsymbol{\delta}^{s_1}_A,\ldots,\boldsymbol{\delta}^{s_m}_A) \in A^u$.

Definition 2.15 (Standard signatures).

A signature Σ is *standard* if $\Sigma(\mathcal{B}) \subseteq \Sigma$.

Given a standard signature Σ, a sort of Σ is called an *equality sort* if Σ includes an equality operator eq_s: $s^2 \to \mathsf{bool}$.

Definition 2.16 (Standard algebras). Given a standard signature Σ, a Σ-algebra A is a *standard algebra* if (i) it is expansion of $\mathbb{B}$; (ii) the equality operator eq_s is interpreted as a partial identity on each equality sort s, $i, e.$, for any two elements of A_s, if they are identical, then the operator at these arguments return tt if it returns anything; and if they are not identical, it returns ff if anything.

Two typical examples of partial identity as an interpretation of eq_s are: (1) total equality, where equality is assumed to be "decidable" at sort s; for example, when $s = \mathsf{nat}$; (2) the case when $s = \mathsf{real}$:

$$\mathsf{eq}^A_{\mathsf{real}}(x, y) \quad = \quad \begin{cases} \uparrow & \text{if } \ x = y \\ \mathsf{ff} & \text{otherwise.} \end{cases}$$

Examples 2.17.

(a) A *standard algebra of naturals* $\mathcal{N}$ is formed by standardising the algebra $\mathcal{N}_0$, with total equality and order operations on $\mathbb{N}$.

algebra	$\mathcal{N}$
import	$\mathcal{N}_0, \quad \mathcal{B}$
functions	$\mathsf{eq}^{\mathcal{N}}_{\mathsf{nat}}, \ \mathsf{less}^{\mathcal{N}}_{\mathsf{nat}}: \quad \mathbb{N}^2 \to \mathbb{B}$

(b) A *standard partial algebra* $\mathcal{R}_p$ *on the reals* is formed similarly by standardising the field $\mathcal{R}_1$ itself a partial algebra, with partial equality and order operations on $\mathbb{R}$:

algebra	$\mathcal{R}_p$
import	$\mathcal{R}_1, \quad \mathcal{B}$
functions	$\mathsf{eq}^{\mathcal{R}}_{\mathsf{real}}, \ \mathsf{less}^{\mathcal{R}}_{\mathsf{real}}: \quad \mathbb{R}^2 \to \mathbb{B}$

where

$$\mathsf{eq}^{\mathcal{R}}_{\mathsf{real}}(x,y) \quad = \quad \begin{cases} \uparrow & \text{if } \ x = y \\ \mathsf{ff} & \text{otherwise.} \end{cases}$$

and

$$\mathsf{less}^{\mathcal{R}}_{\mathsf{real}} \quad = \quad \begin{cases} \mathsf{tt} & \text{if } \ x < y \\ \mathsf{ff} & \text{if } \ x > y \\ \uparrow & \text{if } \ x = y. \end{cases}$$

The significance of these partial equality and order operations, in connection with computability and continuity , is discussed in [TZ04].

Definition 2.18 (N-standard signature). A standard signature Σ is called *N-standard* if it includes (as well as bool) the *numerical sort* nat, and also function symbols for the *standard operations* of *zero*, *successor*, *equality* and *order* on the naturals:

$$\begin{aligned} 0 : \quad & \to \mathsf{nat} \\ \mathsf{S} : \ \mathsf{nat} \quad & \to \mathsf{nat} \\ \mathsf{eq}_{\mathsf{nat}} : \ \mathsf{nat}^2 \quad & \to \mathsf{bool} \\ \mathsf{less}_{\mathsf{nat}} : \ \mathsf{nat}^2 \quad & \to \mathsf{bool}. \end{aligned}$$

Definition 2.19 (N-standard algebra). Given an N-standard signature Σ, a corresponding Σ-algebra A is *N-standard* if the carrier A_{nat} is the set of natural numbers $\mathbb{N}$= {0,1,2,...}, and the standard operations (listed above) have their *standard interpretations* on $\mathbb{N}$.

Definition 2.20 (N-standardization of Σ). The N-*standardization* Σ^N of a standard signature Σ is formed by adjoining the sort nat and the operations 0, S, $\mathsf{eq}_{\mathsf{nat}}$, and $\mathsf{less}_{\mathsf{nat}}$.

Definition 2.21 (N-standardization of A). The N-*standardization* A^N of a standard Σ-algebra A is the Σ^N-algebra formed by adjoining the carrier $\mathbb{N}$ together with certain standard operations to A, thus:

algebra	A^N	
import	A	
carriers	$\mathbb{N}$	
functions	0:	$\rightarrow\mathbb{N}$,
	S:	$\mathbb{N}\rightarrow\mathbb{N}$,
	$\mathsf{eq}_{\mathsf{nat}}$, $\mathsf{less}_{\mathsf{nat}}$:	$\mathbb{N}^2 \rightarrow \mathbb{B}$

In this thesis, we will assume, unless stated otherwise:

Assumption 2.22 (N-Standardness).

All signatures Σ and Σ-algebras A are N-standard.

2.2 Algebras A^* of Signature Σ^*

A standard signature Σ, and standard Σ-algebra A, can be expanded in two stages:
(1°) N-standardize these to form Σ^N and A^N.
(2°) Define, for each sort s of Σ, the carrier ${A_s}^*$ to be the set of *finite sequences* or *array* a^* over A_s of "starred sort" s^*.
The resulting algebras A^* have signature Σ^*, which extends Σ^N by including, for each sort s of Σ, the new starred sorts s^*, and certain new function symbols to read and update arrays.
The significance of arrays for computation is that they provide *finite but unbounded memory.* The reason for introducing starred sorts is the lack of effective coding of finite sequences within abstract algebra in general (unlike the case with $\mathbb{N}$).

2.3 Topological partial algebra

Definition 2.23 (Continuity). Given two topological spaces X and Y, a partial function $f : X \rightharpoonup Y$ is continuous if for every open $V \subseteq Y$, the pre-image

$$f^{-1}[V] =_{df} \{x \in X | x \in \boldsymbol{dom}(f) \text{ and } f(x) \in V\}$$

is open in X.

Definition 2.24 (Topological partial algebra). A topological partial algebra is a partial Σ-algebra with topologies on the carriers such that each of the basic Σ-functions is continuous.

Definition 2.25 (N-standard topological partial algebra). An N-standard

topological partial algebra is a topological partial algebra which is also an N-standard algebra, such that the carriers $\mathbb{B}$ and $\mathbb{N}$ have the discrete topologies.

Examples 2.26. (a) *Discrete algebras*: The standard algebras $\mathcal{B}$ and $\mathcal{N}$ of booleans and naturals respectively are topological (total) algebras under the discrete topology. All functions on them are trivially continuous, since the carriers are discrete.
(b) The *topological partial real algebra* $\mathcal{R}_p$ or its N-standardised version $\mathcal{R}_p^N$, by giving $\mathbb{R}$ its usual topology, and $\mathbb{B}$ and $\mathbb{N}$ the discrete topology. Note that the partial operations $\mathsf{eq}^{\mathcal{R}}$ and $\mathsf{less}^{\mathcal{R}}$ are continuous.
(c) *Partial interval algebras* on the closed interval [0,1] have the form

algebra	$\mathcal{I}_p$
import	$\mathcal{R}_p$
carriers	I
functions	$\mathsf{i}_I : I \to \mathbb{R}$,
	$F_1 : I^{m_1} \to I$
	$\ldots$
	$F_k : I^{m_k} \to I$

where I = [0,1] (with its usual topology), i_I is the embedding of I into $\mathbb{R}$, and F_i: $I^{m_i} \to I$ are continuous partial functions. There are also N-standard versions:

algebra	$\mathcal{I}_p^N$
import	$\mathcal{R}_p^N$
carriers	I
functions	$\mathsf{i}_I : I \to \mathbb{R}$,
	...

(d) The N-*standard total real algebra* $\mathcal{R}_t^N$ is defined by

algebra	$\mathcal{R}_t^N$
import	$\mathcal{R}_0,\ \mathcal{N},\ \mathcal{B}$
functions	$\mathsf{div}_{\mathsf{nat}}^{\mathcal{R}} : \mathbb{R} \times \mathbb{N} \to \mathbb{R},$

Here $\mathcal{R}_0$ is the ring of reals, $\mathbb{N}$ is the standard algebra of naturals and $\mathsf{div}_{\mathsf{nat}}^{\mathcal{R}}$ is division of reals by naturals, defined by:

$$\mathsf{div}_{\mathsf{nat}}^{\mathcal{R}}(x,n) \quad = \quad \begin{cases} x/n & \text{if } \ n \neq 0 \\ 0 & \text{if } \ n = 0 \end{cases}$$

which is total and continuous.

Note that $\mathcal{R}_t^N$ does not contain (total) boolean-valued functions $<$ or $=$ on the reals, since they are not continuous (cf. the partial functions $\mathsf{eq}_{\mathsf{real}}$ and $\mathsf{less}_{\mathsf{real}}$ of $\mathcal{R}_p$).

2.4 Metric Algebra

A particular type of topological algebra is a *metric partial algebra*. This is a many-sorted standard partial algebra with an associated metric:

algebra	A
import	$\mathcal{B},\ \mathcal{R}_p$
carriers	$A_1, \ldots, A_r$
functions	$F_1^A : A^{u_1} \to A_{s_k},$
	...
	$F_k^A : A^{u_k} \to A_{s_k},$
	$d_1^A : A_1^2 \to \mathbb{R},$
	...
	$d_r^A : A_r^2 \to \mathbb{R},$

where $\mathcal{B}$ and $\mathcal{R}_p$ are, respectively, the algebras of boolean and reals, the carriers $A_1, \ldots, A_r$ are metric spaces with metrics $d_1^A,\ldots,d_r^A$, respectively, $F_1, \ldots, F_k$ are Σ-function symbols other than $d_1, \ldots, d_r$, and the (partial) function F_i^A are all continuous with respect to these metrics.

Note that the carriers $\mathbb{B}$ and $\mathbb{N}$ have the *discrete metric*, defined by

$$d(x,y) \quad = \quad \begin{cases} 0 & \text{if } x = y \\ 1 & \text{if } x \neq y \end{cases}$$

which induces the discrete topology.

Chapter 3

While computation on partial algebras; Local uniform *While* approximability

In this chapter we present the first of our two models of abstract computation on $\mathbb{R}$, $\boldsymbol{While}(R_t^N)$ approximability, which is based on the ***While*** programming language [TZ00].

This chapter begin by defining the syntax and semantics of the imperative ***While*** programming language. Most of this is adapted from [TZ00]; we also give some new definitions, such as cumulative exhaustion, which are used later.

3.1 Syntax of $\boldsymbol{While}(\Sigma)$

We define the syntax of the ***While*** programming language over the signature Σ.

Definition 3.1 (Atomic statements). $\boldsymbol{AtSt}(\Sigma)$ is the class of atomic statements

S_{at} ,. . ., defined by:

$$S_{\mathsf{at}} ::= \mathsf{skip} \mid \mathtt{x} := t$$

where $\mathtt{x} := t$ is a *concurrent assignment*, i.e., for some product type u, $\mathtt{x}$ is a tuple of distinct variables of type u and $t : u$.

Definition 3.2 (Statements). $\boldsymbol{Stmt}(\Sigma)$ is the class of *statements* $S, \ldots$, generated by:

$$S ::= S_{\mathsf{at}} \mid S_1 ; S_2 \mid \mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi} \mid \mathsf{while}\ b\ \mathsf{do}\ S_0\ \mathsf{od}$$

Definition 3.3 (Procedures). $\boldsymbol{Proc}(\Sigma)$ is the class of *procedures* $P, Q, \ldots$, which have the form:

$$P \equiv \mathsf{func\ in\ a\ out\ b\ aux\ c\ begin}\ S\ \mathsf{end}$$

where $\mathtt{a}, \mathtt{b}$ and $\mathtt{c}$ are lists tuples of *input variables, output variables* and *auxiliary (or local) variables* respectively, and S is the *body*. Further, we stipulate the following:
(i) $\mathtt{a}, \mathtt{b}$ and $\mathtt{c}$ each consist of distinct variables, and they are pairwise disjoint,
(ii) all variables occurring in S must be among $\mathtt{a}, \mathtt{b}$ or $\mathtt{c}$.

If $\mathtt{a} : u$ and $\mathtt{b} : v$, then P is said to have type $u \to v$, written $P : u \to v$. Its input *type* is u and output *type* is v. We write $\boldsymbol{Proc}(\Sigma)_{u \to v}$ for the class of Σ-procedure of type $u \to v$.

Definition 3.4 (State). For each standard Σ-algebra A, a *state* on A is a family $\langle \sigma_s | s \in Sort(\Sigma) \rangle$ of functions

$$\sigma_s : \boldsymbol{Var}_s \to A_s$$

Let $\boldsymbol{State}(A)$ be the set of states on A, with elements σ,... Note that $\boldsymbol{State}(A)$ is the product of the state spaces $\boldsymbol{State}_s(A)$ for all $s \in \mathrm{Sort}(\Sigma)$.

Let σ be a state over A, $\mathtt{x} \equiv (\mathtt{x}_1, \ldots, \mathtt{x}_n) : u$ and $a = (a_1,\ldots,a_n) \in A^u$(for $n \geq 1$). The *variant* $\sigma\{\mathtt{x}/a\}$ of σ is the state over A formed from σ by replacing its value at $\mathtt{x}_i$ by a_i for $i = 1, ..., n$.

3.2 Semantics of $\boldsymbol{While}(\Sigma)$

3.2.1 Semantics of terms

For $t \in \boldsymbol{Term}_s$, we define the partial function

$$[\![t]\!]^A : \boldsymbol{State}(A) \rightharpoonup A_s$$

where $[\![t]\!]^A\sigma$ is the value of t in A at state σ.

The definition is by structural induction on t:

$$[\![\mathtt{x}]\!]^A\sigma = \sigma(\mathtt{x})$$

$$[\![\mathsf{F}(t_1, \ldots, t_m)]\!]^A\sigma \simeq \begin{cases} \mathsf{F}^A([\![t_1]\!]^A\sigma, \ldots, [\![t_m]\!]^A\sigma) & \text{if } [\![t_i]\!]^A\sigma\downarrow \ (1 \leq i \leq m) \\ \uparrow & \text{otherwise} \end{cases}$$

$$[\![\mathsf{if}\ b\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2\ \mathsf{fi}]\!]^A\sigma \simeq \begin{cases} [\![t_1]\!]^A\sigma & \text{if } [\![b]\!]^A\downarrow\mathtt{tt} \\ [\![t_2]\!]^A\sigma & \text{if } [\![b]\!]^A\downarrow\mathtt{ff} \\ \uparrow & \text{if } [\![b]\!]^A\uparrow. \end{cases}$$

For $m = 0$, *i.e.*, for constant sympbol $\mathsf{c} : \rightarrow s$, $[\![\mathsf{c}]\!]^A = \mathsf{c}_A$.

For a *tuple* of terms $t = (t_1, \ldots, t_m)$, we use the notation

$$[\![t]\!]^A\sigma =_{df} ([\![t_1]\!]^A\sigma, \ldots, [\![t_m]\!]^A\sigma).$$

3.2.2 Algebraic operational semantics

Algebraic operational semantics is a general method for defining the meaning of a statement S used in [TZ88], in a wide class of imperative programming languages, as a partial *state transformation*, *i.e.*, a partial function

$$[\![S]\!]^A : \ \boldsymbol{State}(A) \rightharpoonup \boldsymbol{State}(A).$$

(where "$\rightharpoonup$" denotes a partial function.)

Assume, *firstly*, that (for the language under consideration) there is a class $\boldsymbol{AtSt} \subset \boldsymbol{Stmt}$ of *atomic statements* for which we have a (partial) meaning function

$$\langle\!|S|\!\rangle^A : \ \boldsymbol{State}(A) \rightharpoonup \boldsymbol{State}(A),$$

for $S \in \boldsymbol{AtSt}$, and *secondly*, that we have two functions

$$\begin{aligned} \boldsymbol{First} &: \ \boldsymbol{Stmt} \rightharpoonup \boldsymbol{AtSt} \\ \boldsymbol{Rest}^A &: \ \boldsymbol{Stmt} \times \boldsymbol{State}(A) \rightharpoonup \boldsymbol{Stmt}, \end{aligned}$$

where, for a statement S and state σ, $\boldsymbol{First}(S)$ is an atomic statement which gives the *first* step in the execution of S (in any state), and $\boldsymbol{Rest}^A(S, \sigma)$ is a statement which gives the *rest* of the execution in state σ.

Then, we define the "one-step computation of S at σ" function

$$\boldsymbol{Comp}_1^A : \ \boldsymbol{Stmt} \times \boldsymbol{State}(A) \rightharpoonup \boldsymbol{State}(A)$$

by

$$\boldsymbol{Comp}_1^A(S, \sigma) \ \simeq \ \langle\!|\boldsymbol{First}(S)|\!\rangle^A \sigma.$$

Finally, the definition of *the computation step* function

$$\boldsymbol{Comp}^A : \ \boldsymbol{Stmt} \times \boldsymbol{State}(A) \times \mathbb{N} \simeq \boldsymbol{State}(A) \cup \{*\}$$

follows by a simple recursion on n:

$$\begin{aligned} \boldsymbol{Comp}^A(S,\sigma,0) &= \sigma \\ \boldsymbol{Comp}^A(S,\sigma,n+1) &\simeq \begin{cases} * & \text{if } n>0 \text{ and } S \text{ is atomic} \\ \boldsymbol{Comp}^A(\boldsymbol{Rest}^A(S,\sigma), \boldsymbol{Comp}_1^A(S,\sigma), n) & \\ & \text{otherwise.} \end{cases} \end{aligned}$$

Note that for $n=1$, this yields

$$\boldsymbol{Comp}^A(S,\sigma,1) \simeq \boldsymbol{Comp}_1^A(S,\sigma).$$

The symbol '$*$' indicates that the computation is over.

If we put $\sigma_n = \boldsymbol{Comp}^A(S,\sigma,n)$, then the sequence of states

$$\sigma = \sigma_0,\ \sigma_1,\ \sigma_2,\ \ldots,\ \sigma_n,\ \ldots$$

is called the *computation sequence generated by* S at σ. There are three possibilities:

(a) the sequence terminates in a final state σ_l, where $\boldsymbol{Comp}^A(S,\sigma,l+1) = *$;

(b) it is infinite (*global divergence*);

(c) it is undefined from some point on (*local divergence*).

In case (a) the computation has an output, given by the final state; in case (b) the computation is non-terminating, and has no output; and in case (c) the computation is also non-terminating, and has no output, because a state at one of the time cycles is undefined, as a result of a divergent computation of a term.

Now, we are ready to derive the *i/o* (*input/output*) *semantics*. First we define the *length of a computation* of a statement S, starting in state σ, as the partial function

$$\boldsymbol{CompLength}^A : \boldsymbol{Stmt} \times \boldsymbol{State}(A) \rightharpoonup \mathbb{N}$$

by

$$\boldsymbol{CompLength}^A(S,\sigma) \;=\; \begin{cases} \text{least } n \text{ s.t.} & \boldsymbol{Comp}^A(S,\sigma,n+1) \;=\; * \\ & \text{if such an } n \text{ exists} \\ \uparrow & \text{otherwise.} \end{cases}$$

Note that $\boldsymbol{CompLength}^A(S,\sigma)\!\downarrow$ in case (a) above only. Then we define

$$[\![S]\!]^A(\sigma) \;\simeq\; \boldsymbol{Comp}^A(S,\sigma,\boldsymbol{CompLength}^A(S,\sigma)).$$

3.2.3 Operational semantics of statements

We now apply the above theory to the language $\boldsymbol{While}(\Sigma)$.

There are two atomic statements: skip and *concurrent assignment*. We define $\langle\!|S|\!\rangle^A$ for these:

$$\begin{aligned} \langle\!|\mathsf{skip}|\!\rangle^A\sigma &= \sigma \\ \langle\!|\mathtt{x} := t|\!\rangle^A\sigma &= \sigma\{\mathtt{x}/[\![t]\!]^A\sigma\} \end{aligned}$$

Next we define $\boldsymbol{First}$ and $\boldsymbol{Rest}^A$ by structural induction on $S \in \boldsymbol{Stmt}$.

Case 1. S is atomic.

$$\begin{aligned} \boldsymbol{First}(S) &= S \\ \boldsymbol{Rest}^A(S,\sigma) &= \mathsf{skip}. \end{aligned}$$

Case 2. $S \;\equiv\; S_1; S_2$.

$$\begin{aligned} \boldsymbol{First}(S) &= \boldsymbol{First}(S_1) \\ \boldsymbol{Rest}^A(S,\sigma) &\simeq \begin{cases} S_2 & \text{if } S_1 \text{ is atomic} \\ \boldsymbol{Rest}^A(S_1,\sigma); S_2 & \text{otherwise.} \end{cases} \end{aligned}$$

Case 3. $S \equiv$ if b then S_1 else S_2 fi.

$$\begin{aligned} \boldsymbol{First}(S) &= \mathsf{skip} \\ \boldsymbol{Rest}^{A}(S,\sigma) &\simeq \begin{cases} S_1 & \text{if } [\![b]\!]^A\sigma = \mathsf{tt} \\ S_2 & \text{if } [\![b]\!]^A\sigma = \mathsf{ff} \\ \uparrow & \text{if } [\![b]\!]^A\sigma\uparrow. \end{cases} \end{aligned}$$

Case 4. while b do S_0 od.

$$\begin{aligned} \boldsymbol{First}(S) &= \mathsf{skip} \\ \boldsymbol{Rest}^{A}(S,\sigma) &\simeq \begin{cases} S_0;\mathrm{S} & \text{if } [\![b]\!]^A\sigma \downarrow \mathsf{tt} \\ S_2 & \text{if } [\![b]\!]^A\sigma \downarrow \mathsf{ff} \\ \uparrow & \text{if } [\![b]\!]^A\sigma\uparrow. \end{cases} \end{aligned}$$

Lemma 3.5

(i) For S *atomic* $[\![S]\!]^A \simeq \langle\!|S|\!\rangle^A$, *i.e.*,

$$[\![\mathsf{skip}]\!]^A\sigma = \sigma$$

$$[\![\mathtt{x} := t]\!]^A\sigma \simeq \sigma\{\mathtt{x}/[\![t]\!]^A\sigma\}$$

(ii)

$$[\![S_1; S_2]\!]^A\sigma \simeq [\![S_2]\!]^A([\![S_1]\!]^A\sigma).$$

(iii)

$$[\![\text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi}]\!]^A\sigma \simeq \begin{cases} [\![S_1]\!]^A\sigma & \text{if } [\![b]\!]^A\downarrow\mathsf{tt} \\ [\![S_2]\!]^A\sigma & \text{if } [\![b]\!]^A\downarrow\mathsf{ff} \\ \uparrow & \text{if } [\![b]\!]^A\uparrow. \end{cases}$$

(iv)

$$[\![\mathsf{while}\ b\ \mathsf{do}\ S_0\ \mathsf{od}]\!]^A\sigma \quad \simeq \quad \begin{cases} [\![S; \mathsf{while}\ b\ \mathsf{do}\ S\ \mathsf{od}]\!]^A\sigma & \text{if } [\![b]\!]^A\downarrow \mathtt{tt} \\ \sigma & \text{if } [\![b]\!]^A\downarrow \mathtt{ff} \\ \uparrow & \text{if } [\![b]\!]^A\uparrow. \end{cases}$$

Proof. As in [TZ00, § 4.2], adapted to partial algebras.

3.2.4 *While* computability

We can now give the semantics of ***While*** programs. If

$$P \equiv \ \mathsf{in\ a\ out\ b\ aux\ c\ begin}\ S\ \mathsf{end}$$

is a procedure of type $u \to v$, then its meaning is a function

$$P^A : A^u \ \to \ A^v$$

defined as follows. Suppose $\mathsf{a}:\ u$, $\mathsf{b}\colon\ v$, and $\mathsf{c}:\ w$, let σ be any state such that $\sigma[\mathsf{a}]{=}a$, $\sigma[\mathsf{b}]{=}\delta^v$, and $\sigma[\mathsf{c}]{=}\delta^w$ (by the Instantiation Assumption 2.12).

Then

$$P^A(a) \quad \simeq \quad \begin{cases} \sigma'[\mathsf{b}] & \text{if } [\![S]\!]^A\sigma \downarrow \sigma' \\ \uparrow & \text{if } [\![S]\!]^A\sigma \uparrow. \end{cases}$$

Note that P^A is well defined by the Functionality Lemma [TZ00], which also applied to partial algebra [Xie04, Lemma 3.14].

Definition 3.6 (*While* computable function).

(i) A function f on A is said to be *computable* on A by a ***While*** procedure P if $f = P^A$. It is ***While*** *computable* on A if it is computable on A by some ***While*** procedure.
(ii) ***While***(A) is the class of functions ***While*** *computable* on A.

Definition 3.7 (Halting set). The halting set of a procedure $P : u \to v$ on A is the set:

$$Halt^A(P) =_{df} \{a \in A^u | P^A(a) \downarrow\}$$

Definition 3.8 (*While* semicomputable set). A set $R \subseteq A^u$ is ***While*** semicomputable on A if it is the halting set on A of some ***While*** procedure.

3.3 Local uniform *While* approximations on $\mathbb{R}$

3.3.1 Exhaustions; local approximability and continuity

Definition 3.9 (Open exhaustion). Let U be an open subset of $\mathbb{R}$, and (U_n) a sequence of open subset of $\mathbb{R}$, such that $\bigcup_{n=1}^{\infty} U_n = U$. Then (U_n) is called an *open exhaustion* of U.

Definition 3.10 (Effective exhaustion). The open exhaustion (U_n) is effective if for all n, $U_n = (a_n, b_n)$, where a_n,$b_n \in \mathbb{Q}$, and the map $n \mapsto \ulcorner a_n \urcorner$, $n \mapsto \ulcorner b_n \urcorner$ (where $\ulcorner x \urcorner$ denotes the Gödel number of x) are recursive.

Definition 3.11 (Cumulative exhaustion). Let (U_n) be an open exhaustion of U, and let $W_n = \bigcup_{i=1}^{n} U_i$. Then (W_n) is called the *cumulative exhaustion* of U w.r.t (U_n).

Definition 3.12 (Uniform local continuity) . Given a partial function f: $\mathbb{R} \rightharpoonup \mathbb{R}$, and $U = \boldsymbol{dom}(f)$, we say f is *uniformly locally continuous w.r.t* an exhaustion (U_n) of U iff $\forall\ n$, $\forall \varepsilon > 0$, $\exists\ \delta > 0$, $\forall x, y \in U_n$

$$|x - y| < \delta \Rightarrow |f(x) - f(y)| < \varepsilon$$

Definition 3.13 (Effective local uniform continuity)

Given a partial function f: $\mathbb{R} \rightharpoonup \mathbb{R}$, and $U = \boldsymbol{dom}(f)$, we say f is *effectively uniformly locally continuous* w.r.t an effective exhaustion (U_n) of U iff there is a recursive function M: $\mathbb{N}^2 \to \mathbb{N}$ such that:

for all n,k and all $x, y \in U_n$

$$|x - y| < 2^{-M(n,k)} \ \Rightarrow\ |f(x) - f(y)| < 2^{-k}$$

Definition 3.14 (Effective local uniformly approximability)

Given a partial function f: $\mathbb{R} \rightharpoonup \mathbb{R}$ and a sequence of functions g_m: $\mathbb{R} \rightharpoonup \mathbb{R}$ ($m = 0, 1, 2, 3, ...$), $\boldsymbol{dom}(g_m) \supseteq U = \boldsymbol{dom}(f)$ and an effective exhaustion (U_n) of U, we say that g_n *effectively locally uniformly approximates* f w.r.t (U_n) iff

there is a recursive function M: $\mathbb{N}^2 \to \mathbb{N}$ such that

for all m, n, k and $\forall\ x \in W_n = \bigcup_{i=1}^{n} U_i$,

$$m > M(n, k)\ \Rightarrow\ |g_m(x) - f(x)| < 2^{-k}$$

Lemma 3.15. If g_n approximates f effectively locally uniformly on U w.r.t an effective exhaustion (U_n) of U, and each g_n is continuous, then so is f.

Proof. This is a standard result of real analysis [Rud76].

3.3.2 Effectively local uniform *While** approximability

Definition 3.16 (Effective local uniform *While* approximability).
(a) Let $\mathcal{R}$ be a total or partial metric algebra on $\mathbb{R}$ (*e.g* $\mathcal{R}_t^N$ or $\mathcal{R}_p^N$) with signature Σ. Given a partial function f: $\mathbb{R} \rightharpoonup \mathbb{R}$, $U{=}dom(f)$ and an effective exhaustion (U_n) of U, we say f is *effectively locally uniformly* ***While***$(\mathcal{R})$ *approximable* w.r.t (U_n) if there is a ***While***(Σ^N) procedure P : $\mathsf{nat} \times \mathsf{real} \to \mathsf{real}$ such that (i) $\forall$ n, P_n is total on U, and (ii) the sequence P_n effectively locally uniformly approximates f w.r.t (U_n).
(b)Effective local uniform ***While**** *approximability* is defined analogously.

Lemma 3.17. If $f : \mathbb{R} \to \mathbb{R}$ is effectively locally uniformly ***While**** approximable on its domain w.r.t an effective exhaustion (U_n) of U, then f is continuous on U.

Proof. By Lemma 3.16 and the Continuity Theorem for ***While*** computability [TZ00].

Chapter 4

Polynomial and multipolynomial approximability; GL-computability

In this Chapter, we give some important definitions, such as Weierstrass approximability, GL-computability, effective local polynomial approximability and effective local multipolynomial approximability. We give the theorem that connects these models and ***While***$^{(*)}$ approximability presented in Chapter 3, and we also illustrate polynomial and multipolynomial approximability using Maple 9.5.

4.1 Weierstrass, polynomial and multipolynomial approximability

In order to speak of effective Weierstrass approximability, $i, e.$ effective approximability by a sequence of terms, we need some terminology in connection with the *effective representation of term evaluation.*

Let $\boldsymbol{Term}_{\mathtt{x}}(\Sigma)$ be the class of all Σ-terms with variables among the variable list $x \equiv x_1,\ldots, x_n$, and let $\boldsymbol{Term}_{\mathtt{x},s}(\Sigma)$ be the class of such terms of sort s. The term evaluation representing function on A relative to x is the function

$$\boldsymbol{te}^{\,A}_{\mathtt{x},s} : \ulcorner \boldsymbol{Term}_{\mathtt{x},s}(\Sigma)\urcorner \times A^u \ \rightarrow \ A_s$$

(where $\ulcorner S\urcorner$ denotes the set of Gödel numbers of the set S) defined by

$$\boldsymbol{te}^{A}_{\mathtt{x},s} : \ (\ulcorner t\urcorner, \ a_1, \ \ldots, \ a_n) = \ [\![t]\!]^A(\sigma)$$

where $\ulcorner t\urcorner$ is the Gödel numbers of t, and σ is any state on A such that $\sigma(\mathtt{x}_i) = a_i$ $(i = 1,\ldots,n)$.

Definition 4.1 (TEP). The algebra A is said to have the term evaluation property (TEP) if for all $\mathtt{x}$ and and s, the term evaluation respresenting function $\boldsymbol{te}^{A}_{\mathtt{x},s}$ is ***While*** computable on A.

Many well-known varieties (*i.e.*, equationally axiomatizable classes of algebras) have the TEP; for example, semigroups, groups and rings, as well as $\mathcal{R}^N_t$.

Definition 4.2 (Effective local Weierstrass approximability)
Given a partial function f: $\mathbb{R}\rightharpoonup\mathbb{R}$, U=$\boldsymbol{dom}(f)$ and an effective exhaustion (U_n) of U, we say that f is *effectively locally* Σ-*Weierstrass approximable w.r.t* (U_n) if,
for some $\mathtt{x} : u$, there is a total computable function

$$h : \mathbb{N} \rightarrow \ulcorner \boldsymbol{Term}_{\mathtt{x}}(\Sigma)\urcorner,$$

such that, putting $g_n(a) =_{df} te_{\mathtt{x}}(h(n), a)$, the sequence g_n approximates f effectively locally uniformly w.r.t (U_n).

Lemma 4.3. Given a partial function f: $\mathbb{R} \rightharpoonup \mathbb{R}$, $U = \boldsymbol{dom}(f)$) and an effective exhaustion (U_n) of U, f is said to be *effectively locally* Σ-*Weierstrass approximable* w.r.t (U_n) iff it is *effectively locally* Σ^*-*Weierstrass approximable on* U *w.r.t* (U_n).

Proof: This follows from the $\Sigma^*/\ \Sigma$ Conservativity Theorem [TZ00,§ 3.15], which states that every Σ^*-term of a sort in Σ, all of whose variables are also of sorts in Σ only, can be effectively transformed to a semantically equivalent Σ-term. □

We shall therefore speak of "effective local Weierstrass approximability" to mean effective local Σ or Σ^*-Weierstrass approximability.

Let $\boldsymbol{Poly}_{\mathtt{x}}$ be the set of polynomial expressions in $\mathtt{x}$ with rational coefficients. **Definition 4.4 ($\mathbb{Q}$-polynomial definability on $\mathbb{R}$).** A function $f : \mathbb{R} \to \mathbb{R}$ is $\mathbb{Q}$-polynomially definable on $\mathbb{R}$ if it is explicitly definable by a term in $\boldsymbol{Poly}_{\mathtt{x}}$.

Lemma 4.5 (Equivalence of explicit and $\mathbb{Q}$-polynomial definability on $\mathbb{R}$). A $\Sigma(\mathcal{R}_t^N)$-*term of sort* real *can be effectively transformed to a semantically equivalent* $\mathbb{Q}$-*polynomial.*

Proof. Briefly: we eliminate all occurrences of the "if" operator in the term, using totality of R_t^N and connectedness of $\mathbb{R}$ [TZ00, §9]. The result can easily be expressed as a $\mathbb{Q}$-polynomial.

Definition 4.6 (Effective local $\mathbb{Q}$-polynomial approximability). Let

$$\boldsymbol{val}_{\mathtt{x}} : \ulcorner \boldsymbol{Poly}_{\mathtt{x}} \urcorner \times \mathbb{R} \to \mathbb{R}$$

be the standard evaluation of $\boldsymbol{Poly}_{\mathtt{x}}$ in $\mathbb{R}$.

Given a partial function f: $\mathbb{R} \rightharpoonup \mathbb{R}$, $U = \boldsymbol{dom}(f)$, and an effective exhaustion (U_n) of U, we say f is *effectively locally polynomially approximable on U w.r.t (U_n)* if there is a total computable function

$$h : \mathbb{N} \to \ulcorner \boldsymbol{Poly}_{\mathtt{x}} \urcorner$$

such that, putting $g_n(x) =_{df} \boldsymbol{val}_{\mathtt{x}}(h(n), x)$, the sequence g_n approximates f effectively locally uniformly w.r.t (U_n).

Lemma 4.7 (Equivalence of Weierstrass and $\mathbb{Q}$-polynomial approximability on $\mathbb{R}$). Effective local Wererstrass approximbility over $\mathcal{R}_t^N$ corresponds to effective local $\mathbb{Q}$-polynomial approximability on $\mathbb{R}$.

Proof: By Lemma 4.5. □

Definition 4.8 ($\mathbb{Q}$-multipolynomial). Given a finite sequence of $\mathbb{Q}$-polynomials $(p_1, p_2, \ldots, p_n)$ and a sequence of disjoint intervals $(U_1, \ldots, U_n)$, we can define a $\mathbb{Q}$-multipolynomial $q(x)$ with domain $\bigcup_{i=1}^{n} U_i$ as follows:

$$q(x) \quad = \quad \begin{cases} p_1(x) & \text{if } \ x \in U_1 \\ p_2(x) & \text{if } \ x \in U_2 \\ \ldots & \\ p_n(x) & \text{if } \ x \in U_n \end{cases}$$

We denote $q = [p_1 \restriction U_1, \ldots, p_n \restriction U_n]$.

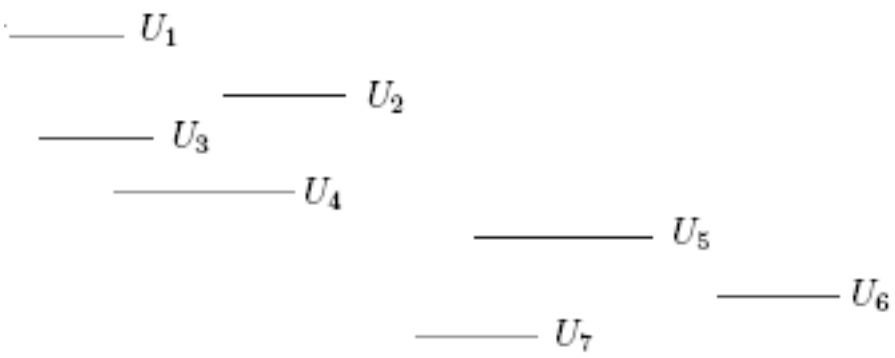
Definition 4.9 (Partition). Let (U_n) be the effective exhaustion of U. Let (W_n) be the corresponding cumulative exhaustion of U. $W_n = \bigcup_{i=1}^{n} U_i$. Note that the U_i are not in general disjoint. However W_n can be represented uniquely as the union of a disjoint sequence of open intervals:

$$W_n = \bigcup_{i=1}^{l_n} V_i^n, \qquad (l_n \leq n), \qquad (*)$$

where each V_i^n is the union of some U_j $(1 \leq j \leq n)$

Then, (*) is called the *partition* of W_n w.r.t (U_n).

Example 4.10 (Partition). We give a example of partition of a cumulative exhaustion (W_n) of U w.r.t (U_n):



The partition of W_n w.r.t (U_n) is as follows:

$$W_1 = U_1 = V_1^1$$

$$W_2 = U_1 \cup U_2 = V_1^2 \cup V_2^2,$$

$$W_3 = U_1 \cup U_2 \cup U_3 = V_1^3 \cup V_2^3.$$

$$W_4 = U_1 \cup U_2 \cup U_3 \cup U_4 = V_1^4$$

$$W_5 = U_1 \cup U_2 \cup U_3 \cup U_4 \cup U_5 = V_1^5 \cup V_2^5$$

$$W_6 = U_1 \cup U_2 \cup U_3 \cup U_4 \cup U_5 \cup U_6 = V_1^6 \cup V_2^6 \cup V_3^6$$

$$W_7 = U_1 \cup U_2 \cup U_3 \cup U_4 \cup U_5 \cup U_6 \cup U_7 = V_1^7 \cup V_2^7 \cup V_3^7$$

Definition 4.11 (Effective local $\mathbb{Q}$-multipolynomial approximability). Given a partial function $f : \mathbb{R} \rightharpoonup \mathbb{R}$, $U = \boldsymbol{dom}(f)$ and an effectively exhaustion (U_n) of U. Let (W_n) be the cumulative exhaustion of U determined by (U_n), with partition $W_n = \bigcup_{i=0}^{l_n} V_i^n$, $l_n \leq n$ on W_n. Suppose the computable sequence of $\mathbb{Q}$-multipolynomials (q_n) *converges effectively locally uniformly to* f w.r.t (W_n), then, we say f is effectively locally $\mathbb{Q}$-multipolynomially approximable w.r.t (U_n).

Lemma 4.12 (Equivalence of Weierstrass and multipolynomial approximability). Given a partial function $f : \mathbb{R} \rightharpoonup \mathbb{R}$, $U = \boldsymbol{dom}(f)$ and an effectively exhaustion (U_n) of U. Then, effective local Weierstrass approximability of f over $\mathcal{R}_t^N$ w.r.t an effective exhaustion (U_n) is equivalent to effective local $\mathbb{Q}$-multipolynomial approximability of f w.r.t (U_n).

Proof. Let (W_n) be the cumulative exhaustion of U determined by (U_n), with partition $W_n = \bigcup_{i=1}^{l_n} V_i^n$, $l_n \leq n$. Using the connectedness of V_i^n, and because $W_n \subseteq W_{n+1} \subseteq U$, effective local Weierstrass approximability over $\mathcal{R}_t^N$ w.r.t (W_n) is equivalent to effective local $\mathbb{Q}$-multipolynomial approximability w.r.t (W_n) (using Lemma 4.7). We can conclude: effective local Weierstrass approximability over $\mathcal{R}_t^N$ w.r.t (U_n) of U is equivalent to effective local $\mathbb{Q}$-multipolynomial approximability w.r.t (U_n) . □

Definition 4.13 (BCP).

A total Σ-algebra A has the *boolean computability property (BCP)* if for any closed Σ-boolean term b, its value b^A (=tt or ff, by totality) can be effectively computed by a total recursive function

$$f : \ulcorner \boldsymbol{CT}_{\mathsf{bool}}(\Sigma) \urcorner \rightarrow \mathbb{B}$$

where $f(\ulcorner b \urcorner) = b^A$ (Note that $\boldsymbol{CT}_{\mathsf{bool}}(\Sigma)$ is the set of closed boolean Σ- terms).

Example 4.14. $\mathcal{R}_t^N$ has both the TEP and the BCP.

Lemma 4.15. Suppose A has the TEP. Given variables $\mathtt{x} : u$, let

$$h : \mathbb{N} \rightarrow \ulcorner \boldsymbol{Term}_{\mathtt{x},s}(\Sigma) \urcorner$$

be a total recursive function. Then there is a $\boldsymbol{While}(\Sigma)$ procedure P: $\mathsf{nat} \times u \rightarrow s$ such that for all $\mathtt{x} \in A^u$ and $n \in \mathbb{N}$,

$$P^A(n, \mathtt{x}) = \boldsymbol{te}^A_{\mathtt{x},s}(h(n), \mathtt{x})$$

where $A = R_t^N$.

Lemma 4.16. Let $P : \mathsf{nat} \times \mathbb{R} \to \mathbb{R}$ be a ***While***$^{(*)}$ procedure over R which defines a function

$$P^A : \mathbb{N} \times \mathbb{R} \rightharpoonup \mathbb{R}$$

where $A{=}R_t^N$. Given an effective exhaustion (U_n) of U. $U \supseteq \boldsymbol{dom}(P^A)$. Then there is a total recursive function h: $\mathbb{N} \to \ulcorner \boldsymbol{Term}_{\mathtt{x}}(\Sigma) \urcorner$ such that for all $a \in U$ and $n \in \mathbb{N}$

$$\boldsymbol{te}_{\mathtt{x}}(h(n), a) = P^A(n, a).$$

Proof. Let W_n be the cumulative exhaustion of U determined by (U_n). The proof use (i) connectedness of the components of W_n and the totality of R_t^N to show that any boolean test gives a constant value (true or false) independent of the state, and (ii) the BCP to effectively decide such a test by evaluating a closed instance of the boolean term $b(\mathtt{x})$, formed by replacing the real variable $\mathtt{x}$ by a rational in the relevant interval V, since by connectedness of V and continuity of $b(\mathtt{x})$, its value must be constant over V.

Lemma 4.17. Given a partial function f: $\mathbb{R} \rightharpoonup \mathbb{R}$, $U{=}\boldsymbol{dom}(f)$ and an open exhaustion (U_n) of U. Suppose f is *effectively locally uniformly* w.r.t (U_n). Then the following are equivalent:

(i) f is *effectively locally uniformly* ***While***$^{(*)}(\mathcal{R}_t^N)$ *approximable* w.r.t. (U_n);

(ii) f is *effectively locally* $\mathbb{Q}$*-multipolynomially approximable* w.r.t. (U_n);

Proof. Consider the assertion

(*) f is *effectively locally Weierstrass approximable* w.r.t (U_n)

From Lemma 4.15 and 4.16, we know $(i) \Leftrightarrow$ (*), From Lemma 4.12, we know $(ii) \Leftrightarrow$ (*). The result follows. □

4.2 GL-computability and Equivalence Theorem

4.2.1 GL-computability

Definition 4.18 (Computable sequence of reals). A sequence of real numbers (x_n) is *computable* (as a sequence) if there is a computable double sequence of rationals (r_{nk}) such that, for all k and n:

$$|r_{nk} - x_n| \leq 2^{-k}$$

Definition 4.19 (Sequential computability). Given a partial function $f\colon \mathbb{R} \rightharpoonup \mathbb{R}$, $U = \boldsymbol{dom}(f)$, we say f is *sequentially computable* on U iff:
f maps every computable sequence of reals $x_n \in U$ into a computable sequence $(f(x_n))$ of reals.

Definition 4.20 (GL-computability).
Given a partial function $f\colon \mathbb{R} \rightharpoonup \mathbb{R}$, $U = \boldsymbol{dom}(f)$ and an effective exhaustion (U_n) of U, we say f is GL-computable w.r.t. (U_n) iff:
(i) f is *sequentially computable* on U, and
(ii) f is *effectively locally uniformly continuous* w.r.t. (U_n).

Note that we use the definition of GL-computability in [PER89].

Remark 4.21. Suppose (U_n) is an effective open exhaustion of $U = \boldsymbol{dom}(f)$, and f is *not effectively locally uniformly continuous* w.r.t (U_n). Then we can often define a

refined exhaustion $(U_n^{'})$ of U. such that:

(i) f is *effectively locally uniformly continuous* w.r.t $(U_n^{'})$;

(ii) $\forall\ n\ \exists\ m$, $\overline{U_n^{'}} \subset U_m$. (where $\overline{U_n^{'}}$ is the topological closure of $U_n^{'}$)

Suppose $U_n = (a_n, b_n)$, Then, define:

$$U_{n,k} = (a_n + \frac{1}{k},\ b_n - \frac{1}{k}),$$

now consider the double array:

$$U_0^0,\ U_0^1,\ U_0^2,\ \ldots$$

$$U_1^0,\ U_1^1,\ U_1^2,\ \ldots$$

$$U_2^0,\ U_2^1,\ U_2^2,\ \ldots$$

$$\ldots,\ \ldots,\ \ldots,\ \ldots$$

Then form a list, e.g by enumerating along the diagonals of the above array,

$$U_0^{'} = U_0^0,$$

$$U_1^{'} = U_1^0,$$

$$U_2^{'} = U_0^1,$$

$$U_3^{'} = U_2^0,$$

$$U_4^{'} = U_1^1,$$

$$U_5^{'} = U_0^2,$$

$$\ldots$$

Then $(U_n^{'})$ is also an exhaustion of U, which is a refinement of (U_n). Furthermore f is *effectively locally uniformly continuous* w.r.t $(U_n^{'})$.

Example 4.22. The function $f(x)$=tan x is *continuous* on U but not *effectively locally uniformly continuous* w.r.t (U_n) where,

$$U = \mathbb{R}\backslash\{(k+\frac{1}{2})\pi \mid k = 0,\ \pm 1,\ \pm 2,\ \ \ldots\}$$

and

$$U_n = ((n-\frac{1}{2})\pi,\ (n+\frac{1}{2})\pi).$$

Now we can give the refined exhaustion $U_{n,k}$ of U, where:

$$U_{n,k} = ((n-\frac{1}{2})\pi + \frac{1}{k},\ \ (n+\frac{1}{2})\pi - \frac{1}{k}).$$

Define $(U_n^{'})$ from $(U_{n,k})$ as in Remark 4.21. Then, tan x is *effectively locally uniformly continuous* w.r.t $(U_n^{'})$.

Lemma 4.23. Given a partial function f: $\mathbb{R}\rightharpoonup\mathbb{R}$, U=$\boldsymbol{dom}(f)$ and an exhaustion (U_n) of U. Suppose f is effectively locally uniformly continuous w.r.t (U_n). Then the following are equivalent:

(i) f is effectively locally multipolynomially approximable w.r.t (U_n),

(ii) f is GL-computable w.r.t (U_n),

(iii) f is effectively locally polynomially approximable w.r.t (U_n).

Proof:

Firstly we prove $(ii) \Rightarrow (i)$.

Let W_n be the cumulative exhaustion of U determined by (U_n), with the partition:

$$W_n = \bigcup_{i=0}^{l_n} V_i^n, \qquad \text{for some } l_n \leq n.$$

Put V_i^n=(a_i, b_i). Then for i= 1, ..., l_n, $a_{i-1} < b_{i-1} < a_i < b_i$. let

$$J_n = \overline{W_n} = \bigcup_{i=0}^{l_n} [a_i, b_i].$$

We modify the proof in [PER89] that for a function f with $[a, b]$

GL-computability $\Rightarrow$ *polynomial approximability.*

For each n, define a multi-polynomials q_n on J_n=$\overline{W_n}$ such that, for all $x \in J_n$:

$$|f(x) - q_n(x)| \leq 2^{-n}$$

Note that for $i = 1, \ldots, l_n$,

$$\overline{V_i^n} \subseteq \overline{W_n}$$

and construct a polynomial p_i^n, for all $x \in V_i^n$ s.t

$$|f(x) - p_i^n(x)| \leq 2^{-n}$$

Define $q_n = [p_1^n, \ldots, p_{l_n}^n]$ on $\overline{W_n}$. Note that by condition (ii) of Remark 4.21, we can assume that for some $m > n$,

$$\overline{W_n} \subseteq W_m \subseteq U. \quad (*)$$

Then f is *effectively locally multi-polynomially approximable* on each W_k by (q_n), and so, f is *effectively locally multipolynomially approximable* w.r.t (U_n).

Secondly we prove $(i) \Rightarrow (ii)$.

Clearly, the multipolynomials are a GL-computable sequence w.r.t (U_n) . Applying Theorem 4 of ([PER89], Chapter 0), which proves closure of GL-computable under effective uniform convergence, we can show f is GL-computable on U w.r.t (U_n).

Now we prove the equivalence of (iii) and (i).

The implication $(iii) \Rightarrow (i)$ is trivial.

Next we prove $(i) \Rightarrow (iii)$, so assume (i),since $(i) \Rightarrow (ii)$ (proved above), we know f is GL-computable w.r.t (U_n).

Let W_n be the cumulative exhaustion of U determined by (U_n),

$$W_n = \bigcup_{i=0}^{l_n} V_i^n, \quad \text{for some } l_n \leq n.$$

$$J_n = \overline{W_n} = \bigcup_{i=0}^{l_n} [a_i, b_i]$$

Now, we let:

$$K_n = CH(J_n) = [a_0, b_{l_n}]$$

where $CH(J_n)$ is the convex hull of J_n.

Then we can construct a new function f_n on K_n as follows, let

$$f_n(x) \quad = \quad \begin{cases} f(x) & \text{if } \ x \in J_n \\ \frac{(f(a_{i+1})-f(b_i))(x-b_i)}{a_{i+1}-b_i} + f(b_i) & \text{if } \ x \in [b_i, a_{i+1}], (i = 0 \ldots l_n - 1). \end{cases}$$

This means that $f_n(x)$ is the liner interpolation of the points of (b_i, a_{i+1}) on $[b_i, a_{i+1}]$, $(i = 0 \ldots l_n - 1)$. Since f_n is GL-computable on K_n= [a_0 , b_{l_n}], by the Effective Weierstrass Theorem of ([PER89], Chapter0) we know f_n is effectively locally polynomially approximable on K_n = [a_0,b_{l_n}] by a sequence of polynomials (p_n).

Because

$$W_n \ \subseteq \ J_n \ \subseteq \ K_n,$$

we conclude that f is effectively locally polynomially approximable on each W_k by (p_n). By (*) above, for some $m > n$, $\overline{W_n} \subseteq W_m \subseteq U$. We conclude that f is effectively locally polynomially approximable by (p_n) w.r.t (U_n).

This complete the proof. □

4.2.2 The Equivalence Theorem for partial functions; Examples.

Now we give the equivalence Theorem of partial functions for the total abstract model $\boldsymbol{While}^*(R_t^N)$, the concrete model GL($\mathbb{R}$), and the polynomial and multipolynomial approximation models.

Theorem 1. Given a partial function f: $\mathbb{R} \rightharpoonup \mathbb{R}$, U=$\boldsymbol{dom}(f)$ and an exhaustion (U_n) of U. Suppose f is *effectively locally uniformly continuous* w.r.t (U_n). Then the following are equivalent:

(i) f is *effectively locally uniformly* $\boldsymbol{While}^{(*)}(R_t^N)$ *approximable* w.r.t. (U_n);

(ii) f is *effectively locally multipolynomially approximable* w.r.t. (U_n);

(iii) f is *effectively locally polynomially approximable* w.r.t. (U_n);

(iv) f is GL-*computable* w.r.t.(U_n).

Proof:

By Lemma 4.17, we have the equivalence of the first two assertions.

By Lemma 4.23, we have the equivalence of (ii),(iii) and (iv).

Examples 4.24. (1) Consider to the function $f(x) = \frac{1}{x}$, U=(-1,0) ∪ (0,1).

$$U_1 = V_1 = (-1, -1/n),$$

$$U_2 = V_2 = (1/n, 1)$$

$$W_n = (-1, -1/n) \bigcup (1/n, 1),$$

$$I_n = [-1, -1/n] \bigcup [1/n, 1]$$

$$K_n = [-1, 1]$$

In Figure 1, we show the multipolynomial q_n which approximate f for n=5.

In Figure 2, we show the polynomial p_n which approximate f for n=5.

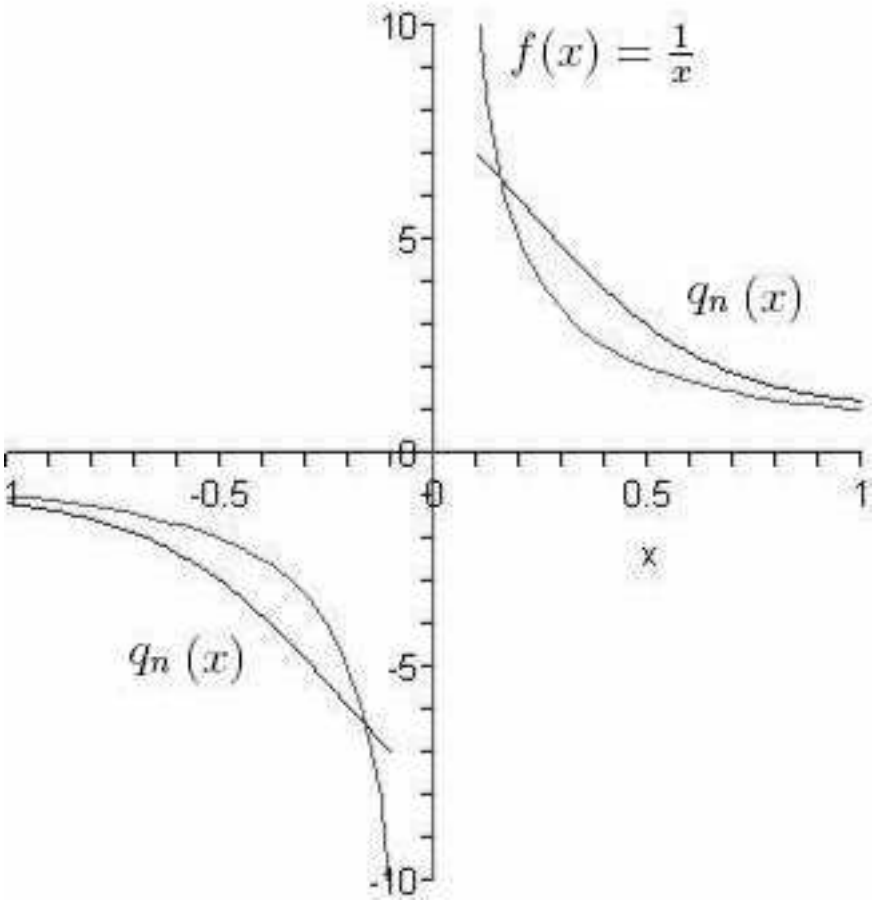


Figure 1: Multipolynomial approximation for $\frac{1}{x}$ (n=5)

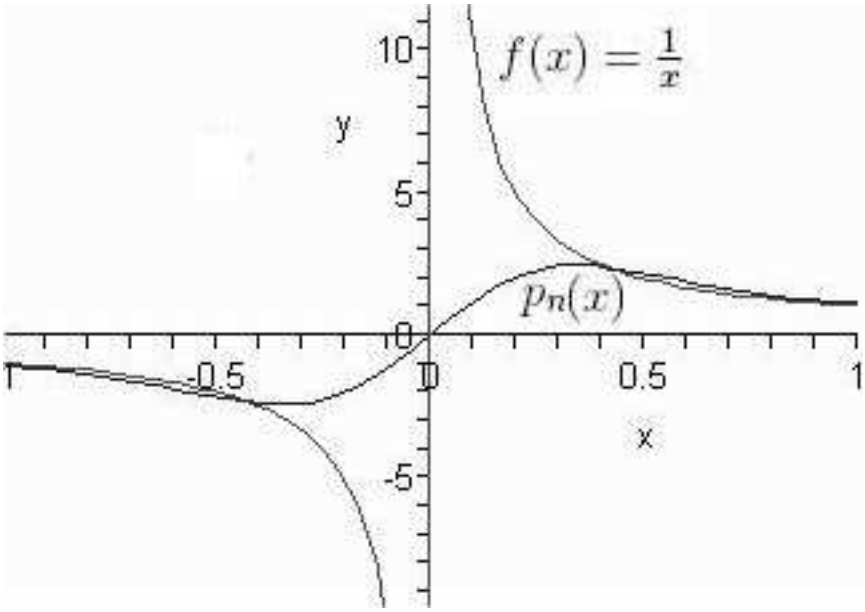


Figure 2: polynomial approximation for $\frac{1}{x}$ (n=5)

(2) f=tan$(x * \frac{\pi}{2})$,

$$U = \mathbb{R}\backslash\{2k+1 \mid k = 0,\ \pm 1,\ \pm 2,\ \ \ldots\}$$

The multipolynomial q_n which approximate f for n=3, (on W_3) is shown in Figure 3.

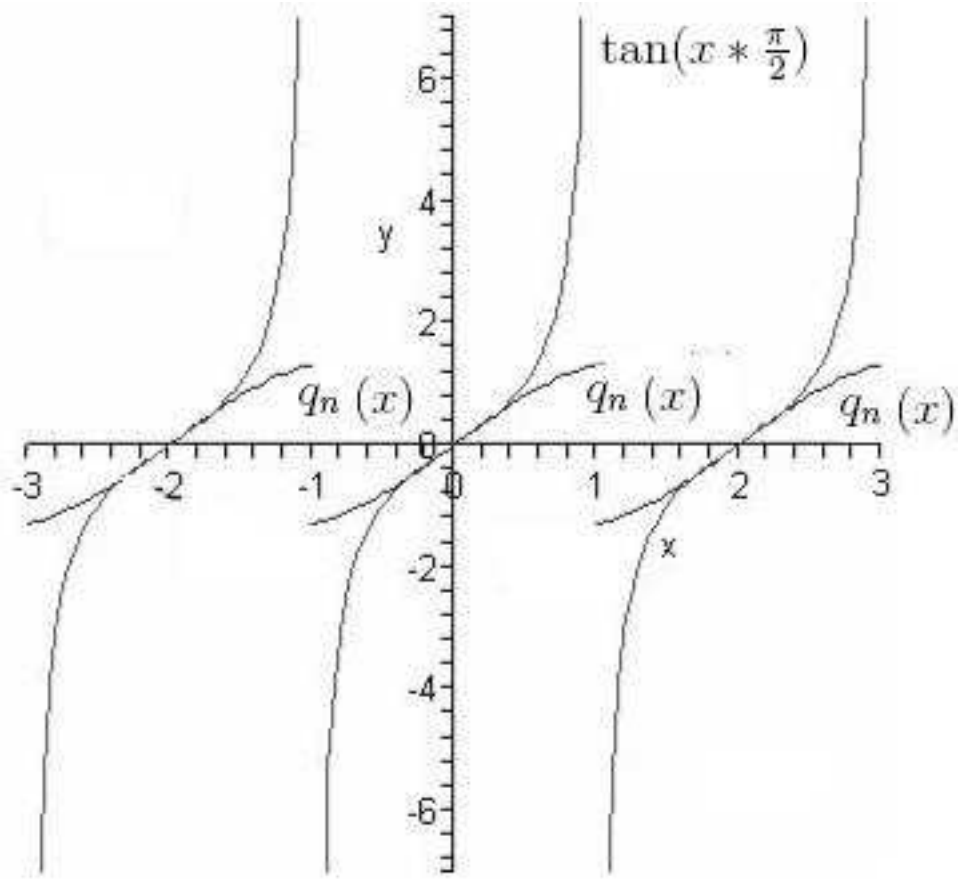


Figure 3: Multipolynomial approximation for $\tan(x * \frac{\pi}{2})$ (n=3)

Remark 4.25.

(1) It can be seen that multipolynomials give a much better approximation than polynomials.

(2) We used the polynomial approximation sequences (p_n) defined in [PER89]:

$$p_n(x) = \frac{1}{C_n} \int_{-\frac{M}{2}}^{\frac{M}{2}} P_n(t-x) f(t) dt,$$

where

$$C_n = \int_{-M}^{M} P_n(x)dx$$

from which we also define the multipolynomial sequences (q_n). (Note that the interval $[-\frac{M}{2}, \frac{M}{2}]$ in the definition of p_n must be replaced by the appropriate closed intervals in these examples.)

It would be interesting to investigate whether other polynomial sequences commonly used in approximation theory would give better results.

Chapter 5

While programming with countable choice

The programming language $\boldsymbol{WhileCC}(\Sigma)$ is an extension of $\boldsymbol{While}(\Sigma)$ with an extra "choose" rule of term formation. It will form the basic of the second of our two abstract computation model on $\mathbb{R}$ ($\boldsymbol{WhileCC}^*(\mathbb{R}_p)$ approximability). Here, we give the complete definition of its syntax and semantics from [TZ05].

5.1 Syntax of $\boldsymbol{WhileCC}(\Sigma)$

There are four syntactic classes: *variables,terms, statements, and procedures*
(*a*) $\boldsymbol{Var}(\Sigma)$ is the class of Σ-program variables, and for each Σ-sort s, $\boldsymbol{Var}_s$ is the class of program variables of sort s: $\mathsf{a}^s, b^s, \ldots, \mathtt{x}^s, \mathtt{y}^s, \ldots$,
(b) $\boldsymbol{PTerm}(\Sigma)$ is the class of Σ-program terms t, ..., and for each Σ-sort s, $\boldsymbol{PTerm}_s$ is the class of program of sort s. These are generated by the rules

$$t ::= \mathtt{x}^s \mid F(t_1, \ \ldots, \ t_n) \mid \mathsf{if}\ b\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2\ \mathsf{fi} \ \mid \ \mathsf{choose}\ \mathsf{z}^{\mathsf{nat}} : b$$

where s, s_1, ..., s_n are Σ-sorts, F: $s_1 \times \ldots \times s_n \to s$ is a Σ-function symbol, $t_i \in$

$\boldsymbol{PTerm}_s$ for $i = 1, \ldots, n (n \geq 0)$, and b is a boolean term.

The "choose" term has sort nat. Think of "choose" as a generalization of the *constructive least number operator* least z : b which has the value k in case $b[\mathsf{z}/\mathtt{k}]$ is true and $b[\mathsf{z}/i]$ is defined and false for all $i < k$, and is undefined in case no such k exists.

Here "choose z : b " selects some value k such that b$[\mathsf{z}/k]$ is true, if any such k exists (and is undefined otherwise). In the abstract semantics [TZ05], the meaning is the set of all possible k's (hence "countable choice"). Any concrete model will select a particular k, according to the implementation.

Note that the program terms extend the algebraic terms ($i, e.$ the terms over the signature Σ) by including in their construction the "choose" operator, which is not an operation of Σ. An alternative formulation would have "choose" *not* as part of the term consruction, but rather as a new atomic program statement:"choose z : b".

We write $t : s$ to indicate that $t \in \boldsymbol{PTerm}_s$, and for $u = s_1 \times \ldots \times s_m$, we write $t : u$ to indicate that t is a u-tuple of program terms, *i.e.*, a tuple of program terms of sorts $s_1, \ldots, s_m$. We also use the notation b for boolean terms.

Definition 5.1 (Atomic statements) $\boldsymbol{AtSt}(\Sigma)$ is the class of atomic statements S_{at} ,. . . ,defined by:

$$S_{\mathsf{at}} ::= \mathsf{skip} \mid \mathsf{div} \mid \mathsf{x} := t$$

where x:=t is a concurrent assignment and div stands for "divergence".

$\boldsymbol{Stmt}(\Sigma)$ and $\boldsymbol{Proc}(\Sigma)$ of statements and procedures respective are defined as before (Def 3.2, 3.3), relative to the new definition of program terms and atomic statements (Def 5.1).

5.2 Semantics of $\boldsymbol{WhileCC}(\Sigma)$

5.2.1 Notation

(a) $\mathcal{P}_\omega(X)$ is the set of all countable subsets of a set X, including the empty set.

(b) $\mathcal{P}^+_\omega(X)$ is the set of all countable non-empty subsets of X.

(c) We write $Y^\uparrow$ for $Y \cup \{\uparrow\}$,where '$\uparrow$' denotes divergence.

(d) We write $f\colon X \rightrightarrows Y$ for $f\colon X \to \mathcal{P}_\omega(Y)$

(e) We write $f\colon X \rightrightarrows^+ Y$ for $f\colon X \to \mathcal{P}^+_\omega(X)$

We adapt and extend the algebraic operational semantics given in Chapter 3.

5.2.2 Details

(a) Semantics of program terms. The meaning of $t \in \boldsymbol{PTerm}_s$ is a function

$$[\![t]\!]^A : \boldsymbol{State}(A) \rightrightarrows^+ A^\uparrow_s.$$

The definition is by structural induction on t:

$$\begin{aligned}
[\![\mathsf{x}]\!]^A\sigma &= \{\sigma(\mathsf{x})\} \\
[\![\mathsf{c}]\!]^A\sigma &= \{\mathsf{c}^A\} \\
[\![\mathsf{F}(t_1,\ldots,t_m)]\!]^A\sigma &= \{y \mid \exists x_1 \in A \cap [\![t_1]\!]\sigma \ldots \exists x_m \in A \cap [\![t_m]\!]\sigma : \mathsf{F}^A(x_1,\ldots,x_m) \downarrow y\} \\
&\quad \cup \{y \mid \exists x_1 \in A \cap [\![t_1]\!]\sigma \ldots \exists x_m \in A \cap [\![t_m]\!]\sigma : \mathsf{F}^A(x_1,\ldots,x_m) \uparrow\} \\
&\quad \cup \{\uparrow \in [\![t_i]\!]^A\sigma \text{ for some } i, 1 \le i \le m\} \\
[\![\mathsf{if}\ b\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2\ \mathsf{fi}]\!]^A\sigma &= \{y \mid (\mathsf{tt} \in [\![b]\!]^A\sigma \wedge y \in [\![t_1]\!]^A\sigma) \vee (\mathsf{ff} \in [\![b]\!]^A\sigma \wedge y \in [\![t_2]\!]^A\sigma) \\
&\quad \cup \{\uparrow \mid \uparrow \in [\![b]\!]^A\sigma\} \\
[\![\mathsf{choose}\ \mathsf{z} : b]\!]^A &= \{\, n \in \mathbb{N} \mid \mathsf{tt} \in [\![b]\!]^A\sigma\{\mathsf{z}/n\}\} \\
&\quad \cup \{\uparrow \mid \forall n \in \mathbb{N}(\mathsf{ff} \in [\![b]\!]^A\sigma\{\mathsf{z}/n\} \vee \uparrow \in [\![b]\!]^A\sigma\{\mathsf{z}/n\})\}
\end{aligned}$$

Notice that $[\![\mathsf{choose\ z} : b]\!]^A$ could include both natural numbers and "$\uparrow$" , since for any n, $[\![b]\!]^A\sigma\{\mathsf{z}/n\}$ could include both $\mathtt{tt}$ and $\mathtt{ff}$.

(b) Semantics of atomic statements The meaning of $S_{\mathsf{at}} \in \boldsymbol{AtSt}$ is a function

$$\langle\!|S_{\mathsf{at}}|\!\rangle : \ \boldsymbol{State}(A) \rightrightarrows^+ \boldsymbol{State}(A)^\uparrow$$

defined by:

$$\begin{aligned}
\langle\!|\mathsf{skip}|\!\rangle^A\sigma &= \{\sigma\} \\
\langle\!|\mathsf{div}|\!\rangle^A\sigma &= \{\uparrow\} \\
\langle\!|\mathtt{x} := t|\!\rangle^A\sigma &= \{\sigma\{\mathtt{x}/a\} | a \in A \cap [\![t]\!]^A\sigma\} \cup \{\uparrow \mid \uparrow \in [\![t]\!]^A\sigma\}
\end{aligned}$$

(c) The ***First*** and ***Rest*** operations. The operation

$$\boldsymbol{First} : \boldsymbol{Stmt} \ \rightarrow \boldsymbol{AtSt}$$

is defined as in Chapter 3 (§ 3.2.2), namely:

$$\boldsymbol{First}(S) \quad = \quad \begin{cases} S & \text{if } \ S \text{ is atomic} \\ \boldsymbol{First}(S_1) & \text{if } \ S \equiv S_1; S_2 \\ \mathsf{skip} & \text{otherwise.} \end{cases}$$

The operation:

$$\boldsymbol{Rest}^A : \ \boldsymbol{Stmt} \times \boldsymbol{State}(A) \ \rightrightarrows^+ \ \boldsymbol{Stmt}$$

is defined as follows:

Case 1. S is atomic.

$$\boldsymbol{Rest}^A(S, \sigma) \quad = \quad \{\mathsf{skip}\}.$$

Case 2. $S \equiv S_1; S_2$.

Case 2a. S_1 is atomic. Then $\boldsymbol{Rest}^A(S,\sigma) = \{S_2\}$

Case 2b. S_1 is not atomic. Then $\boldsymbol{Rest}^A(S,\sigma) =$

$$\{S'; S_2 | S' \in \boldsymbol{Rest}^A(S_1,\sigma)\} \cup \{\mathsf{div} | \mathsf{div} \in \boldsymbol{Rest}^A(S_1,\sigma)\}$$

Case 3. $S \equiv$ if b then S_1 else S_2 fi. Then $\boldsymbol{Rest}^A(S,\sigma)$ contains all of

$$\begin{cases} S_1 & \text{if } \mathsf{tt} \in [\![b]\!]^A\sigma \\ S_2 & \text{if } \mathsf{ff} \in [\![b]\!]^A\sigma \\ \mathsf{div} & \text{if } \uparrow \in [\![b]\!]^A\sigma. \end{cases}$$

Case 4. $S \equiv$ while b do S_0 od. Then $\boldsymbol{Rest}^A(S,\sigma)$ contains all of

$$\begin{cases} S_0; S & \text{if } \mathsf{tt} \in [\![b]\!]^A\sigma \\ \mathsf{skip} & \text{if } \mathsf{ff} \in [\![b]\!]^A\sigma \\ \mathsf{div} & \text{if } \uparrow \in [\![b]\!]^A\sigma. \end{cases}$$

Note in *Case 3* and *4*, more than one condition may hold.

(*d*) *Computation step.*

From ***First*** we can define the computation step function

$$\boldsymbol{CompStep}^A : \boldsymbol{Stmt} \times \boldsymbol{State}(A) \rightrightarrows^+ \boldsymbol{State}(A)^\uparrow$$

which is like the one-step computation function $\boldsymbol{Comp}_1^A$ of § 3.2.2 except for being multi-valued:

$$\boldsymbol{CompStep}^A(S,\sigma) = \langle\!|\boldsymbol{First}(S)|\!\rangle^A\sigma.$$

(*e*) *The computation tree.*

The computation sequence, which is basic to the semantics of ***While*** computations Chapter 3 (§ 3.2.2), is replaced here by a computation tree

$$\boldsymbol{CompTree}^A(S,\sigma)$$

of a statement S at a state σ. This is an ω-branching tree, branching according to all possible outcomes (i,e, "output states") of the one-step computation function $\boldsymbol{CompStep}^A$. Each node of this tree is labelled by either a state or '$\uparrow$'
Any actual ("concrete") computation of statement S at state σ corresponds to one of the paths through this tree. The possibilities for any such path are:
(i) it is finite, ending in a leaf containing a state: the final state of the computation;
(ii) it is finite, ending in a leaf containing '$\uparrow$' (local divergence);
(iii) it is infinite (global divergence).

Correspondingly, the function $\boldsymbol{Comp}^A$ (§ 3.2.2) is replaced by a computation tree stage function :

$$\boldsymbol{CompTreeStage}^A : \ \boldsymbol{Stmt} \times \boldsymbol{State}(A) \times \mathbb{N} \ \rightrightarrows^+ \ (\boldsymbol{State}(A)^\uparrow)^{<\omega}$$

defined by a simple recursion on n:

$$\begin{aligned} \boldsymbol{CompTreeStage}^A(S,\sigma,0) &= \{\sigma\} \\ \boldsymbol{CompTreeStage}^A(S,\sigma,n+1) &= \begin{cases} \{\sigma'\} \text{ if } n>0 \text{ and } S \text{ is atomic} \\ \boldsymbol{CompTreeStage}^A(S',\sigma',n)^\uparrow \end{cases} \end{aligned}$$

where $S' \in \boldsymbol{Rest}^A(\mathrm{S},\sigma)$ and $\sigma' \in \boldsymbol{CompStep}^A(S,\sigma)$
Then $\boldsymbol{CompTree}^A(S,\sigma)$ is defined as the "limit" over n of $\boldsymbol{CompTreeStage}^A(S,\sigma,n)$.

(f) *Semantics of statements*

From the semantic computation tree we can easily define the i/o semantics of statements

$$[\![S]\!]^A : \ \boldsymbol{State}(A) \rightrightarrows^+ \boldsymbol{State}(A)^{\uparrow}.$$

Namely,

> $[\![S]\!]^A\sigma$ is the set of states and/or '↑' at all leaves in $\boldsymbol{CompTree}^A(S,\sigma)$, together with '↑' if $\boldsymbol{CompTree}^A(S,\sigma)$ has an infinite path.

Note that, by its definition, $[\![S]\!]^A\sigma$ cannot be empty. It will contain (at least) '↑' if there is at least one computation sequence leading to divergence, *i.e*, a path of the computation tree which is either infinite or ends in a '↑' leaf.

(g) *Semantics of procedures*

Finally if

$$P \equiv \ \mathsf{in\ a\ out\ b\ aux\ c\ begin}\ S\ \mathsf{end}$$

is a procedure of type $u \to v$, then its meaning in A is a function

$$P^A : \ A^u \ \to \ \ A^{v\uparrow}$$

defined as follows. For $x \in A^u$,

$$P^A(x) = \{\sigma^{'}(b) | \sigma^{'} \in [\![S]\!]^A\sigma\} \cup \{\uparrow \mid \uparrow \in [\![S]\!]^A\sigma\}$$

where σ is any state on A such that σ[a]=x.

Definition 5.4. (a) A many-valued function $f : A^u \rightrightarrows^+ A_s^{\uparrow}$ is $\boldsymbol{WhileCC}$ computable on A if there is a $\boldsymbol{WhileCC}$ procedure P such that $f = P^A$.

(b) A partial function $f : A^u \longrightarrow A_s$ is $\boldsymbol{WhileCC}$ computable on A if there is a deterministic $\boldsymbol{WhileCC}$ procedure P: $u \to s$ such that for all $x \in A^u$,

(i) $f(x) \downarrow y \Longrightarrow P^A(x)=\{y\}$, and
$(ii) f(x) \uparrow \Longrightarrow P^A(x) =\{\uparrow\}$

Remark 5.5 The semantics for ***WhileCC*** procedures is given by countably many-valued functions. If we were to start with algebras with many-valued basic operations, as in [Bra96, Bra99], the algebraic operational semantics could handle this just as easily, by adapting the clause for the the basic Σ-function f in part (a) of the semantic definition (§ 5.2.2).

5.3 The language $\boldsymbol{WhileCC}^*(\Sigma)$

The language $\boldsymbol{While}^*(\Sigma)$ is formed by augmenting ***While*** with auxiliary array variables. The importance of $\boldsymbol{While}^*$ computability lies in the fact that it forms the basis for a generalised Church-Turing Thesis for computability on abstract many-sorted algebras [TZ00, §8].

Here, similarly, the language $\boldsymbol{WhileCC}^*(\Sigma)$, which can be viewed as $\boldsymbol{WhileCC}(\Sigma)$ augmented by auxiliary array variables (or as $\boldsymbol{While}^*(\Sigma)$ augmented by the 'choose' construct).

More precisely, a $\boldsymbol{WhileCC}^*(\Sigma)$ procedure is a $\boldsymbol{WhileCC}(\Sigma^*)$ procedure in which the input and output variables have sorts in Σ only. (However the auxiliary variables may have starred sorts.)

Thus it defines a countably-many-valued function on any N-standard Σ-algebra.

Theorem 5.6 For any total Σ-algebra A and f: $A^u \to A_s$,

f is $\boldsymbol{WhileCC}^*$ computable over $A \Longleftrightarrow f$ is partial recursive over A.

This Theorem have been proved in ([TZ04],§4.4).

Corollary 5.7 For any f: $\mathbb{N}^m \to \mathbb{N}$

f is ***WhileCC**** computable over $\mathbb{N} \Longleftrightarrow f$ is partial recursive over $\mathbb{N}$.

5.4 Approximable *WhileCC** computability

The basic notion of computability that we will be using in working with metric algebras is not so much computability, as rather approximable computability on metric algebras, as discussed in ([TZ99],§ 9). Here, we give the definition to the nondeterministic case with countable choice.

Let A be a metric Σ-algebra, u a Σ-product type and s a Σ-type. Let

$$P : \mathsf{nat} \times u \to s$$

be a ***WhileCC****(Σ). Put

$$P_n^A =_{df} P^A(n, .) : A^u \rightrightarrows^+ A_s^{\uparrow}$$

Note that for all $x \in A^u$, $P_n^A(x) \neq \phi$

Definition 5.8 (*WhileCC** approximability to a single-valued function).

Let f: $A^u \rightharpoonup A_s$ be a single-valued partial function on A.

(a) f is ***WhileCC**** approximable by P on A if for all $n \in \mathbb{N}$ and all $x \in A^u$:

$$x \in \boldsymbol{dom}(f) \Rightarrow \uparrow \notin P_n^A(x) \subseteq B(f(x), 2^{-n}) \quad (1)$$

(b) f is strictly ***WhileCC**** approximable by P on A if in addtion to (1)

$$x \notin \boldsymbol{dom}(f) \Rightarrow P_n^A(x) = \{\uparrow\}.$$

Definition 5.9 (*WhileCC** approximability to a many-valued function).

Let $f\colon A^u \rightrightarrows A_s$ be a countably-many-valued function on A.

(a) f is ***WhileCC**** approximable by P on A if for all $n \in \mathbb{N}$ and all $x \in A^u$

$$f(x) \neq \phi \;\Rightarrow \uparrow \notin P_n^A(x) \subseteq \bigcup_{y \in f(x)} B(y, 2^{-n})$$

and

$$f(x) \subseteq \bigcup_{y \in P_n^A(x)} B(y, 2^{-n})$$

(b) f is strictly ***WhileCC**** approximable by P on A if in addition,

$$f(x) = \phi \;\Rightarrow\; P_n^A(x) = \{\uparrow\}.$$

Some examples are given in [TZ04, §5], which interested reader may consult.

Chapter 6

Tracking computability and equivalence theorem

In this chapter, we present our final (concrete) model based on tracking functions, and give the theorem which connects all the models.

6.1 Tracking computability for partial functions

Let A be an N-standard metric algebra. Let X be a family $\langle X_s | s \in Sort(\Sigma)\rangle$ of subsets $X_s \subseteq A_s$. Each X_s can be viewed as a metric subspace of the metric space A_s.

Definition 6.1 (enumeration). An *enumeration* of X is a family

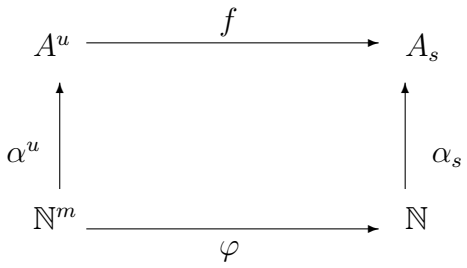
$$\alpha = \langle \alpha_s : \Omega_s \twoheadrightarrow X_s | s \in sort(\Sigma)\rangle$$

of surjective maps $\alpha_s : \Omega_s \twoheadrightarrow X_s$ for some family $\Omega = \langle \Omega_s | s \in \boldsymbol{Sort}(\Sigma)\rangle$ of sets $\Omega_s \subseteq \mathbb{N}$. The family X is said to be *enumerated* by α. We say that $\alpha : \Omega \twoheadrightarrow X$ is an *enumeration* of X, and call the pair (X, α) an *enumerated* $\boldsymbol{Sort}(\Sigma)$-*family* of subspaces of A.

We also write $\Omega_s = \Omega_{\alpha,s}$ to make explicit the fact that $\Omega_s = \boldsymbol{dom}(\alpha_s)$, and we use the notation $\Omega_\alpha^u = \Omega_{\alpha,s_1} \times \ldots \times \Omega_{\alpha,s_m}$ and $X^u = X_{s_1} \times \ldots \times X_{s_m}$ where $u = s_1 \times \ldots \times s_m$.

Assume now that A is an N-standard metric Σ-algebra and (X, α) is an enumerated Sort(Σ)-family of subspaces of A, with enumeration $\alpha : \Omega \twoheadrightarrow X$.

Definition 6.2 (Tracking functions). Let $f : A^u \rightharpoonup A_s$ and $\varphi : \mathbb{N}^m \longrightarrow \mathbb{N}$. Then φ is α-tracking function for f if the following diagram commutes:



in the sense for all $k = (k_1, \ldots, k_m) \in \mathbb{N}^m$ and writing $\alpha^u(k) = (\alpha_{s_1}(k_1), \ldots, \alpha_{s_m}(k_m))$:

$\varphi(k) \downarrow \;\Rightarrow\; \varphi(k) \in \Omega_{\alpha,s} \wedge f(\alpha^u(k) \downarrow \alpha_s(\varphi(k))$

$\varphi(k) \uparrow \;\Rightarrow\; f(\alpha^u(k)) \uparrow .$

Definition 6.3 (α-computability).
The function $f : A^u \longrightarrow A_s$ is α-computable on $U{=}\boldsymbol{dom}(f)$, if it has a recursive α-tracking function on U.

Definition 6.4 (Enumerated Σ-subalgebra). Let X be a Σ-subalgebra of A. An enumeration α of X, together with a family of tracking functions for its operations, is called an *enumerated Σ-subalgebra of A*.

Definition 6.5 (Σ-effective enumeration). The enumeration α is said to be Σ-

effective if all the basic Σ-functions on A (including the metrics) are α-computable.

Definition 6.6 (Computational closure). Let X be a subspaces of A, enumerated by α. We define a family

$C_\alpha(X) = \langle C_\alpha(X)_s \mid s \in \boldsymbol{Sort}(\Sigma)\rangle$

of sets $C_\alpha(X)_s$ of α-*computable elements* of A_s, *i.e.*, limits in A_s of effectively convergent Cauchy sequences [TZ04] of elements of X_s, so that

$X_s \subseteq C_\alpha(X)_s \subseteq A_s$,

with corresponding enumerations

$\bar{\alpha_s}\Omega_{\bar{\alpha},s} :\twoheadrightarrow C_\alpha(X)_s$,

(where $\twoheadrightarrow$ denotes surjection). Writing $\bar{\alpha} = \langle \bar{\alpha_s} \mid s \in \boldsymbol{Sort}(\Sigma)\rangle$, we call the enumerated subspace $(C_\alpha(X), \bar{\alpha})$ the *computable closure* of (X, α) in A.

6.2 Application to computability on $\mathbb{R}$

Let

$$\alpha_0 : \mathbb{N} \;\to\; \mathbb{Q},$$

be a (fixed, standard) enumeration of the rationals. From this we construct the set

$$C_0 = C_{\alpha_0}(\mathbb{Q}),$$

of *recursive reals*, with enumerations

$$\bar{\alpha_0} : \Omega_0 \;\twoheadrightarrow\; C_0.$$

Note that α_0 is ***While*** computable over $\mathcal{R}_p^N$. $(C_{\alpha_0}(\mathbb{Q}), \bar{\alpha_0})$ is the *computable closure* of $(\mathbb{Q}, \bar{\alpha_0})$ in $\mathbb{R}$. Note that $\bar{\alpha_0}$ is $\Sigma(\mathcal{R}_p^N)$-effective.

6.3 Equivalence Theorem including $\bar{\alpha_0}$-computable

Lemma 6.7 Given a partial function f: $\mathbb{R} \rightharpoonup \mathbb{R}$, U=***dom***(f) and an effective exhaustion (U_n) of U. Then the following are equivalent:

(i) f is $\bar{\alpha_0}$-computable

(ii) f is effectively uniformly $WhileCC^*(\mathcal{R}_p^N)$ approximable.

Proof. This follows from the Completeness Theorem of [TZ04]. □

Lemma 6.8. Given a partial function f: $\mathbb{R} \rightharpoonup \mathbb{R}$, U=***dom***(f) and an effective exhaustion (U_n) of U. Suppose f is effectively locally uniformly continuous w.r.t (U_n). Then the following are equivalent:

(i) f is *GL-computable* w.r.t (U_n),

(ii) f is $\bar{\alpha_0}$-*computable*.

Proof. Firstly we need to prove $(ii) \Rightarrow (i)$. So suppose f is $\bar{\alpha_0}$-computable. We must show f is GL-computable w.r.t (U_n).

According to the definition of GL-computability, we must prove : **(1)** f is sequentially computable, and **(2)** f is effectively locally uniformly continuous w.r.t (U_n).

We first prove **(1)** f is sequentially computable. So take any computable sequence (x_n) on U. We must show that the sequence $(f(x_n))$ is also computable.

Since (x_n) is a computable sequence on U, there is by a recursive function ψ: $\mathbb{N} \to \mathbb{N}$, such that:

$$x_n = \bar{\alpha_0} \circ \psi(n)$$

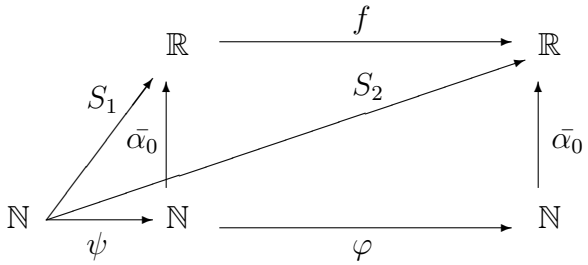
Let S_1=$\bar{\alpha_0} \circ \psi$. Then for all n,

$$x_n = S_1(n).$$

Since f is $\bar{\alpha_0}$-computable, we know there is a $\bar{\alpha_0}$-tracking function $\varphi : \mathbb{N} \rightharpoonup \mathbb{N}$ for f. Put $y_n{=}f(x_n)$. Let $S_2{=}\bar{\alpha_0} \circ \varphi \circ \psi$. Then for all n,

$$y_n = S_2(n).$$

So (y_n) is also a computable sequence.



We have proved (1): f is sequentially computable on U.

In addition by assumption, f is effectively locally uniformly continuous w.r.t (U_n).

Hence f is GL-computable w.r.t (U_n).

Secondly, we will prove $(i) \Rightarrow (ii)$. So, suppose f is GL-computable w.r.t an effective exhaustion (U_n) of U. We must show that f is $\bar{\alpha_0}$-computable; *i.e.* we must construct an $\bar{\alpha_0}$-tracking function φ for f. Let

$$e \in \Omega \;=\; \boldsymbol{dom}(\bar{\alpha_0}).$$

Then for all n, $\{e\}(n)$ is the Gödel number of a rational r_n, so that (r_n) is an effective Cauchy sequence of rationals. Put $s_n = f(r_n)$. Since f is GL-computable, (s_n) is an effective sequence of rationals. Hence there is an index $e^{'}$, found effectively from e, such that for all n,

$$\bar{\alpha_0}(\{e^{'}\}(n)) = s_n.$$

Define $\varphi(e) = e^{'}$. Then φ is partial recursive.

Since, moreover, f is effective uniformly continuous w.r.t (U_n), the sequence (s_n) is an effective Cauchy sequence of rationals.

Let x be the limit of r_n and y be the limit of (s_n). Then x, $y \in C_0$ and $f(x) = y$. So φ is a rational recursive $\bar{\alpha_0}$-tracking function for f, Hence f is $\bar{\alpha_0}$-computable. This complete the proof. □

Note: The Lemma 6.8 was stated without proof in [TZ05].

Theorem 2 Given a partial function f: $\mathbb{R} \rightharpoonup \mathbb{R}$, U=$\boldsymbol{dom}(f)$ and an effective exhaustion (U_n) of U. Suppose f is effectively locally uniformly continuous w.r.t (U_n). Then the following are equivalent:

(i) f is GL-*computable* w.r.t (U_n),

(ii) f is $\bar{\alpha_0}$-*computable*,

(iii) f is *effectively locally* $\mathbb{Q}$-*polynomially approximable* w.r.t (U_n),

(iv) f is *effectively locally* $\mathbb{Q}$-*multipolynomially approximable* w.r.t (U_n),

(v) f is *effectively locally uniformly* $\boldsymbol{While}^{(*)}(\mathcal{R}_t^N)$ *approximable* w.r.t (U_n),

(vi) f is *effectively uniformly* $\boldsymbol{WhileCC}^{(*)}(\mathcal{R}_p^N)$ *approximable* .

Proof.

By Theorem 1, we have the equivalence of (i), (iii) , (iv) and (v).

By Lemma 6.7, we have the equivalence of (ii) and (vi).

By Lemma 6.8, we have the equvialence of (i) and (ii). □

Chapter 7

Conclusion and future work

7.1 Conclusion

In this thesis, we studied six models of computability of partial functions on the real numbers; two abstract, two concrete and two based on polynomial approximation:

(*i*) $\boldsymbol{While}^{(*)}(R_t^N)$ *approximable computability*;

(*ii*) $\boldsymbol{WhileCC}^{(*)}(R_p^N)$ *approximable computability*;

(*iii*) GL-*computability*;

(*iv*) $\bar{\alpha_0}$-*computability*;

(*v*) *Effective local uniform polynomial approximability*;

(*vi*) *Effective local uniform multipolynomial approximability.*

We proved their equivalence under two assumption on the function f: (1) $\boldsymbol{dom}(f)$ is the union of an effective sequence (U_n) of rational open intervals; and (2) f is effectively locally uniformly continuous w.r.t (U_n).

7.2 Future work

We list some future work in this area:

(1) To try to generalize our results to functions with domain of the form:

$$\bigcap_{m=1}^{\infty} \bigcup_{n=1}^{\infty} U_{mn}$$

for an effective double sequence of rational intervals (U_{mn}). This is the most general form of the domain of a TTE concretely computable function [Wei00].

(2) To generalize our result to function with domain $\mathbb{R}^n$ for $n > 1$, and more generally, to metric algebras, such as Banach space.

The problem in generalizing the domain from $\mathbb{R}$ to $\mathbb{R}^n$ is connected with the problem of generalizing the liner interpolation construction of Lemma 4.23 to more than one dimension.

Bibliography

[Her05] Peter. Hertling. A banach-mazur computable but not markov computable function on the computable real numbers. *Annals of Pure and Applied Logic*, 132:227–246, 2005.

[Her06] Peter. Hertling. A sequentially computable function that is not effectively continuous at any point. *Journal of complexity*, 22:752–767, 2006.

[PER89] Marian B. Pour-EI and Jonathan I. Richards. *Computability in Analysis and physics*. Springer-Verlag, 1989.

[Rud76] Walter Rudin. *Principles of Mathmatical analysis*. New york, 1976.

[SHT99] V. Stolenberg-Hansen and J. Tucker. Concrete models of computation for topological algebras. *Theoret.Comput*, 219:347–378, 1999.

[TZ88] J.V. Tucker and J.I. Zucker. Program correctness over abstract data types, with error-state semantics. *CWI Monographs*, 6, 1988.

[TZ99] J.V. Tucker and J.I. Zucker. Computable functions by 'while' programs on topological partial algebras. *Theoretical Computer Science*, 219:379–420, 1999.

[TZ00] J.V. Tucker and J.I. Zucker. Computable functions and semicomputable sets on many-sorted algebras. *Oxford University Press*, 5:317–523, 2000.

[TZ02] J.V. Tucker and J.I. Zucker. Abstract computability and algebraic specification. *ACM Transactions on Computational logic*, 3:279–333, 2002.

[TZ04] J.V. Tucker and J.I. Zucker. Abstract versus concrete computation on metric partial algebras. *ACM Transactions on Computational logical*, 5:611–688, 2004.

[TZ05] J.V. Tucker and J.I. Zucker. Computable total functions on metric algebras,algebraic specifications and dynamical systems. *The Journal of Logic and Algebraic Programming*, 62:71–108, 2005.

[Wei00] Klaus Weihrauch. *Computable Analysis*. Springer-Verlag, 2000.

[Xie04] Bo Xie. *Characterizations of Semicomputable Sets of Real numbers*. 2004.