Abstract versus Concrete Computation on Metric Partial Algebras

J.V. TUCKER
University of Wales, Swansea
and
J.I. ZUCKER
McMaster University

In the theory of computation on topological algebras there is a considerable gap between so-called abstract and concrete models of computation. In concrete models, unlike abstract models, the computations depend on the representation of the algebra. First, we show that with abstract models, one needs algebras with *partial operations*, and computable functions that are both *continuous* and *many-valued*. This many-valuedness is needed even to compute single-valued functions, and so *abstract models must be nondeterministic even to compute deterministic problems*. As an abstract model, we choose the "while'-array programming language, extended with a nondeterministic "countable choice" assignment, called the $\boldsymbol{WhileCC}^*$ model. Using this, we introduce the concept of *approximable many-valued computation* on metric algebras. For our concrete model, we choose metric algebras with *effective representations*. We prove: (1) for any metric algebra A with an effective representation α, $\boldsymbol{WhileCC}^*$ approximability implies computability in α, and (2) also the converse, under certain reasonable conditions on A. From (1) and (2) we derive an equivalence theorem between abstract and concrete computation on metric partial algebras. We give examples of algebras where this equivalence holds.

Categories and Subject Descriptors: F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*Computatability theory*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Computatability theory*; *Proof theory*

General Terms: Theory, Verification

Additional Key Words and Phrases: abstract computation, data types, countable choice, effective Banach space, effective metric space, metric algebra, partial algebra, topological algebra.

1. INTRODUCTION

The theory of data in computer science is based on many sorted algebras and homomorphisms. The theory originates in the 1960s, and has developed a wealth of theoretical concepts, methods and techniques for the specification, construction,

The research of the second author was supported by a grant from the Natural Sciences and Engineering Research Council (Canada) and by a Visiting Fellowship from the Engineering and Physical Sciences Research Council (U.K.)

Authors' addresses: J. V. Tucker, Department of Computer Science, University of Wales, Swansea SA2 8PP, Wales; email: J.V.Tucker@swansea.ac.uk.; J. I. Zucker, Department of Computing and Software, McMaster University, Hamilton, Ontario L8S 4L7, Canada; email: zucker@mcmaster.ca.

and verification of software and hardware systems. It is a significant achievement in computer science and has exerted a profound influence on programming [Goguen et al. 1978; Meseguer and Goguen 1985; Wirsing 1991]. However, given the absolutely fundamental nature of its subject matter — data — there are many fascinating and significant open problems. An important general problem is:

> *To develop a comprehensive theory of specification, computation and reasoning with infinite data.*

By infinite data we mean real numbers, spaces of functions, streams of bits or reals, waveforms, multidimensional graphics objects, video, and analogue and digital interfaces. The application areas are obvious: scientific modelling and simulation, embedded systems, graphics and multimedia communications.

Data types of infinite data are modelled by topological many-sorted algebras. In this paper we consider computability theory on topological algebras and investigate the problem

> *To compare and integrate high-level, representation independent, abstract models of computation with low-level, representation dependent, concrete models of computation in topological algebras.*

Computability theory lies at the technical heart of theories of both specification and reasoning about such systems. There are many disparate ways of defining computable functions on topological algebras and some have (different) significant mathematical theories. In the case of the real numbers one can contrast the approaches in books such as [Aberth 1980; 2001; Pour-El and Richards 1989; Weihrauch 2000; Blum et al. 1998].

Generally speaking, the models of computation for an algebra can be divided into two kinds: the *abstract* and *concrete*.

With an *abstract model of computation* for an algebra, the programs and algorithms do not depend on any representation of the algebra and are invariant under isomorphisms. Abstract models originated in the late 1950s in formalising flowcharts, and include program schemes and many languages that have been used in the study of program semantics [de Bakker 1980; Apt and Olderog 1991]. Examples of such models are the ***While*** programming language over any algebra and the Blum-Cucker-Shub-Smale (BCSS) model [Blum et al. 1989; Blum et al. 1998] over the rings of real or complex numbers. The theory of abstract models is stable: there are many models of computation and the conditions under which they are equivalent are largely known [Tucker and Zucker 1988; 2000]. For example, 'while' programs, flow charts, register machines, Kleene schemes, etc., are equivalent on *any* algebra; the BCSS models are simply instances obtained by choosing the algebra to be a ring or ordered ring.

With a *concrete model of computation* for an algebra the programs and computations are not invariant under isomorphisms, but depend on the choice of a representation of the algebra. To understand invariance requires a study of reductions and equivalences between "computable" representations. Complications arise in relating different concrete representations. Usually, the representations are made from the set $\mathbb{N}$ of natural numbers, and computability on an algebra is reduced to classical computability on $\mathbb{N}$. Concrete models originated in the 1950s, in formalis-

ing the computable functions on real numbers [Grzegorczyk 1955; 1957; Lacombe 1955]. Examples of concrete models are computability via

—effective metric spaces [Moschovakis 1964],
—computable sequence structures [Pour-El and Richards 1989],
—domain representations [Stoltenberg-Hansen and Tucker 1988; 1995; Edalat 1995; 1997],
—type two enumerability [Weihrauch 2000], and
—numbered topological spaces [Spreen 1998; 2001].

The theory of concrete models is not stable, though it seems to be converging: the above models are known to be equivalent under conditions satisfied by many (though not all) important spaces (see [Stoltenberg-Hansen and Tucker 1999] for equivalence results; also [Weihrauch 2000, §9] for counterexamples). In the case of the real numbers, the above concrete models (and others) are all known to be equivalent to Grzegorczyk-Lacombe (GL) computability.

In the theory of computation on algebras, abstract models are implemented by concrete models. Thus, the gap between the models is the gap between high level programming abstractions and low level implementations, and can be explored in terms of the following concepts:

—*Soundness of abstract model*: The functions computable in the abstract model are also computable in the concrete model.
—*Adequacy of abstract model*: The functions computable in the concrete model are computable in the abstract model.
—*Completeness of abstract model*: Functions are computable in the abstract model if, and only if, they are computable in the concrete model.

However, there is a considerable gap between abstract and concrete models of computation, especially over topological data types. For example, the popular abstract model in [Blum et al. 1998] is *not* sound for the main concrete models because of its assumptions about the total computability of relations such as equality. Equality on the real numbers is not everywhere continuous, but in all the concrete models computable functions are continuous (*cf.* Ceitin's Theorem [Ceitin 1959; Moschovakis 1964]). The connection between abstract and concrete models of computation on the real numbers is examined in [Tucker and Zucker 1999] where *approximation* by 'while' programs over a *particular* algebra was shown to be equivalent to the standard concrete model of GL computability over the unit interval.

First attempts at bridging the gap for all topological algebras in general have been made in [Brattka 1996; 1999], using a generalisation of recursion schemes (abstract computability) and Weihrauch's type two enumerability (concrete computability). Here we investigate further the problems in comparing the two classes of models and in establishing a unified and stable theory of computation on topological algebras. We prove new theorems that bridge the gap in the case of computations on metric algebras with partial operations.

By reflecting on a series of examples, we show that to compute functions on topological algebras, it is necessary to consider

(*i*) algebras with partial operations,

(*ii*) computable functions that are both continuous and many-valued, and
(*iii*) approximations by abstract programs.

In particular, *many-valued functions are needed in the abstract model, even to compute single-valued functions*. Thus, to prove an equivalence between abstract and concrete models we must include a nondeterministic construct to define many-valued functions, and in this way use nondeterministic abstract models even to compute deterministic problems. We find that

> *imperative and other abstract programming models must be nondeterministic to express even simple programs on topological data types.*

We choose the ***While*** programming language as an abstract model for computing on any data type, and extend it with the *nondeterministic assignment of countable choice*

$$\mathtt{x} \ :: \ \text{choose } \mathtt{z} : b(\mathtt{z}, \mathtt{x}, \mathtt{y})$$

where $\mathtt{z}$ is a natural number variable and b is a Boolean-valued operation. This new model is called ***WhileCC**** computability ('*CC*' for "countable choice", '*' for array variables.) In particular, we introduce a notion of *approximable many-valued computation*, and formulate and prove the continuity of their semantics. We thus have the partial many-valued functions approximable by a ***WhileCC**** program on A.

As a concrete model, we choose *effective metric spaces*; this is known to be equivalent with several other concrete models. It is an elegant approach, we feel, suitable for theoretical investigation and comparison with other models of computability; some other choices of concrete model (among those listed above) may be closer to practical techniques for exact computation with reals (say).

In computation with effective metric spaces A we pick an enumeration α of a subspace X of A, and construct the subspace $\mathsf{C}_\alpha(X)$ of α-computable elements of A, enumerated by $\overline{\alpha}$. We thus have the partial functions computable on $\mathsf{C}_\alpha(X)$ in the representation $\overline{\alpha}$.

We then prove two theorems that can be summarised (a little loosely) as follows.
Soundness Theorem: *Let A be any metric partial algebra with an effective representation α. Suppose $\mathsf{C}_\alpha(X)$ is a subalgebra of A, effective under $\overline{\alpha}$. Then any function F on A that is **WhileCC*** approximable over A is computable on $\mathsf{C}_\alpha(X)$ in $\overline{\alpha}$.*
This theorem is technically involved but quite general, and gives new insight into the semantics of imperative programs applied to topological data types. The converse theorem is more restricted in its data types:
Adequacy Theorem: *Let A be any metric partial algebra A with an effective representation α. Suppose the representation α is **WhileCC*** computable and dense. Then any function $F : A \to A$ that is computable on $\mathsf{C}_\alpha(X)$ in $\overline{\alpha}$ and effectively locally uniformly continuous in α is **WhileCC*** approximable over A.*

These are combined into a ***Completeness Theorem***. The proper statements of these three theorems are given as Theorems A, B and C (in Sections 8, 9 and 10). Some interesting applications to algebras of real numbers and to Banach spaces are studied.

Here is the structure of the paper. We begin, in Section 2, by explaining the role of partiality, continuity and many-valuedness in computation, using simple examples on the real numbers. In Section 3 we describe topological and metric partial algebras. In Section 4 we introduce the ***WhileCC**** language, give it an algebraic semantics, and define approximable ***WhileCC**** computability. Section 5 is devoted to examples. In Section 6 we prove the continuity of ***WhileCC**** computable many-valued functions. In Section 7 we introduce our concrete model, effective metric spaces, and prove a Soundness Theorem (Theorem A_0) for the special case of surjective enumerations of countable (not necessarily metric) algebras. In Section 8 we define the subspace of elements computable in a metric algebra, and then prove the more general Soundness Theorem (Theorem A) and, in Section 9, the Adequacy Theorem (Theorem B). These are combined into a Completeness Theorem (Theorem C) in Section 10. Concluding remarks are made in Section 11.

This work is part of a research programme — starting in [Tucker and Zucker 1988] and most recently surveyed in [Tucker and Zucker 2000] — on the theory of computability on algebras, and its applications. Specifically, it has developed from our studies of real and complex number computation in [Tucker and Zucker 1992a; 1999; 2000], stream algebras in [Tucker and Zucker 1992b; 1994] and metric algebras in [Tucker and Zucker 2002a].

2. PARTIALITY, CONTINUITY, MANY-VALUEDNESS AND EXTENSIONALITY

When one considers the relation between abstract and concrete models, a number of intriguing problems appear. We explain them by considering a series of examples. Then we formulate our strategy for solving these problems.

Our chosen abstract and concrete models are introduced later (in Sections 4 and 6, respectively), so we must explain the problems of computing on the real number data type in general terms. First, we sketch the abstract and concrete forms of the real number data type. The picture for topological algebras in general will be clear from the examples.

2.1 Abstract versus concrete data types of reals; Continuity; Partiality

(a) Abstract and concrete data types of reals. To compute on the set $\mathbb{R}$ of real numbers with an abstract model of computation, we have only to select an algebra A in which $\mathbb{R}$ is a carrier set. Abstract computability on an algebra A is computability *relative to* A: a function is computable over A if it can be programmed from the operations of A using the programming constructs of the abstract model. Clearly, there are infinitely many choices of operations with which to make an algebra A, and hence there are infinitely many choices of classes of abstractly computable functions. All the classes of abstractly computable functions on $\mathbb{R}$ have decent mathematical theories, resembling the theory of the computable functions on the natural numbers — thanks to the general theory of computable functions on many-sorted algebras [Tucker and Zucker 2000].

In contrast, to compute on $\mathbb{R}$ with a concrete model of computation, we choose a representation map $\alpha : C \to \mathbb{R}$ from a structure C (typically a subset of the naturals $\mathbb{N}$ or Baire space $\mathbb{N}^{\mathbb{N}}$) based on the fact that the reals can be built from the rationals, and hence the naturals, in a variety of equivalent ways (Cauchy sequences, decimal expansions, etc.). Computability of functions on the reals is investigated

using the theory of computable functions on C, applied to $\mathbb{R}$ via α.

To compare concrete with abstract models, we choose an algebra A in which $\mathbb{R}$ is a carrier set and the operations of A are computable with respect to α. For example, multiplication by 3 is not computable in the decimal representation, but the field operations on $\mathbb{R}$ are computable in the Cauchy sequence representation.

In the examples in §2.2 below, we will take as our concrete model the set $\mathbf{CS} \subseteq \mathbb{N}^{\mathbb{N}}$ of fast Cauchy sequences, *i.e.*, sequences (k_n) of naturals such that for all n and all $m > n$, $|r_{k_m} - r_{k_n}| < 2^{-n}$, where $r_0, r_1, r_2, \ldots$ is some standard enumeration of the rationals. Note that the canonical map $\alpha : \mathbf{CS} \to \mathbb{R}$ is continuous and onto.

(*b*) Continuity. Computations with real numbers involve infinite data. The topology of $\mathbb{R}$ defines a process of approximation for infinite data; the functions on the data that are continuous in the topology are exactly the functions that can be approximated to any desired degree of precision.

For abstract models we assume the algebra A that contains $\mathbb{R}$ is a topological algebra, *i.e.*, one in which the basic operations are continuous in its topologies. We expect further that all the computable functions will be continuous. However, the class of functions that can be abstractly computed exactly turns out to be quite limited; *approximate computations* are found to be necessary in abstract models [Tucker and Zucker 1999].

In the concrete models, moreover, it follows from Ceitin's Theorem [Moschovakis 1964] that computable functions are continuous.

Thus, in both abstract and concrete approaches, an analysis of basic concepts shows that computability implies continuity.

(*c*) Partiality. In computing with an abstract model on A we assume A has some boolean-valued functions to test data. For example, in computing on $\mathbb{R}$ we need the functions

$$=_R : \mathbb{R}^2 \to \mathbb{B} \qquad \text{and} \qquad <_R : \mathbb{R}^2 \to \mathbb{B}$$

where $\mathbb{B} = \{\mathtt{tt}, \mathtt{ff}\}$ is the set of booleans. This presents a problem, since total continuous boolean-valued functions on the reals must be constant. Further, as was shown in [Tucker and Zucker 1999], the 'while' and 'while'-array computable functions on connected total topological algebras are precisely the functions explicitly definable by terms over the algebra.

To study the full range of real number computations, we must therefore redefine these tests as *partial* boolean-valued functions, where the undefinedness corresponds intuitively to "divergence" or non-termination of the relevant algorithm. Computation with partial algebras has interesting effects on the theory of computable functions, as indicated in [Tucker and Zucker 1999].

On the basis of these preliminary remarks, we turn to the examples.

2.2 Examples of nondeterminism and many-valuedness

We look at three examples of computing functions on $\mathbb{R}$.

Example 2.2.1 *(Pivot function).* Define the function

$$\mathrm{piv} : \mathbb{R}^n \rightarrow \{1, \ldots, n\}$$

by

$$\mathrm{piv}(x_1,\ldots,x_n) \;=\; \begin{cases} \text{some } i:\ x_i \neq 0 & \text{if such an } i \text{ exists} \\ \uparrow & \text{otherwise.} \end{cases} \tag{1}$$

where '$\uparrow$' denotes "divergence" or undefinedness.

Computation of the pivot is a crucial step in the Gaussian elimination algorithm for inverting matrices.

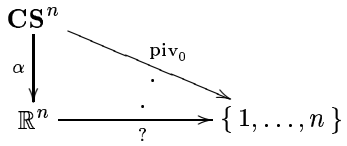
Note that (depending on the precise semantics for the phrase "some i" in (1)) piv is *nondeterministic* or (alternatively) *many-valued* on $\boldsymbol{dom}(\mathrm{piv}) = \mathbb{R}^n\backslash\{0\}$. Further:

(a) There is no *single-valued* function which satisfies the definition (1) and is *continuous* on $\mathbb{R}^n$. For such a function, being continuous and integer-valued, would have to be constant on its domain $\mathbb{R}^n\backslash\{0\}$, with constant value (say) $j \in \{1,\ldots,n\}$. But its value on the x_j-axis would have to be different from j, leading to a contradiction.

(b) However there *is* a computable (and hence continuous!) single-valued function

$$\mathrm{piv}_0 : \mathbf{CS}^n \;\dot{\to}\; \{1,\ldots,n\} \tag{2}$$

with a simple algorithm. (The space $\mathbf{CS}$ was defined in §2.1(a).) Note however that piv_0 is *not extensional* on $\mathbf{CS}^n$ (*i.e.*, not well defined on $\mathbb{R}^n$), or (equivalently) the map (2) cannot be factored through $\mathbb{R}^n$:



In effect, we can regain continuity (for a single-valued function), by foregoing extensionality.

(c) Alternatively, we can maintain continuity *and* extensionality by giving up single-valuedness. For the many-valued function

$$\mathrm{piv}_\omega : \mathbb{R}^n \;\to\; \mathcal{P}_\omega(\{1,\ldots,n\})$$

(where $\mathcal{P}_\omega(\ldots)$ denotes the set of countable subsets of $\ldots$) defined by

$$k \in \mathrm{piv}_\omega(x_1,\ldots,x_n) \iff x_k \neq 0 \qquad (k = 1,\ldots,n)$$

is *extensional* and *continuous*, where a function $f : A \to \mathcal{P}_\omega(B)$ is defined to be continuous iff for all open $Y \subseteq B$, $f^{-1}[Y]$ $(=_{df}\ \{x \in A \mid f(x) \cap Y \neq \emptyset\})$ is open in A. (We will consider continuity of many-valued functions systematically in Section 6.)

Remarks 2.2.2. (a) The many-valued function piv_ω is "tracked" (in a sense to be elucidated in Section 7) by (any implementation of) piv_0.

(b) We could only recover continuity of the piv function by giving up either extensionality (as in (b) above) or single-valuedness (as in (c)).

(*c*) Note however that the complete Gaussian algorithm for inverting matrices is *continuous* and *deterministic* (hence *single-valued*) and *extensional*, even though it contains piv_0 as an essential component!

Example 2.2.3 *("Choose" a rational arbitrarily near a real).* Define a function

$$F\colon \mathbb{R} \times \mathbb{N} \;\to\; \mathbb{N}$$

by

$$F(x,n) \;=\; \text{“some” } k:\ \mathsf{d}(x,r_k) \;<\; 2^{-n} \tag{3}$$

where (as before) $r_0, r_1, r_2, \ldots$ is some standard enumeration of the rationals. Note again:

(*a*) There is no *single-valued, continuous* function F satisfying (3), since such a function, being continuous with discrete range space, would have to be constant in the first argument.

(*b*) But there *is* a single-valued computable (and continuous) function

$$F_0\colon \mathbf{CS} \times \mathbb{N} \;\to\; \mathbb{N}$$

trivially — just define

$$F_0(\xi,n) \;=\; \xi_n.$$

This is, again, *non-extensional* on $\mathbb{R}$.

(*c*) Further, there is a *many-valued, continuous, extensional* function satisfying (3):

$$F_\omega : \mathbb{R} \times \mathbb{N} \;\to\; \mathcal{P}_\omega(\mathbb{N})$$

where

$$F_\omega(x,n) \;=\; \{\, k \mid \mathsf{d}(x,r_k) < 2^{-n} \,\}.$$

Example 2.2.4 *(Finding the root of a function).* (Adapted from [Weihrauch 2000].) Consider the function f_a shown in Figure 1, where a is a real parameter. It is defined by

$$f_a(x) \;=\; \begin{cases} x + a + 2 & \text{if } \; x \le -1 \\ a - x & \text{if } \; -1 \le x \le 1 \\ x + a - 2 & \text{if } \; 1 \le x. \end{cases}$$

This function has either 1 or 3 roots, depending on the size of a. For $a < -1$, f_a has a single (large positive) root; for $a > 1$, f_a has a single (large negative) root; and for $-1 < a < 1$, f_a has three roots, two of which become equal when $a = \pm 1$.

Let g be the (many-valued) function, such that $g(a)$ gives all the non-repeated roots of f_a. This is shown in Figure 2. Again we have the situation of the previous examples:

(*a*) We cannot choose a (single) root of f_a continuously as a function of a.

(*b*) However, one can easily choose and compute a root of f_a continuously as a function of a *Cauchy sequence representation* of a, *i.e.*, non-extensionally in a.

(*c*) Finally, $g(a)$, as a *many-valued* function of a, is continuous. (Note that in order to have continuity, we must exclude the repeated roots of f_a, at $a = \pm 1$.)

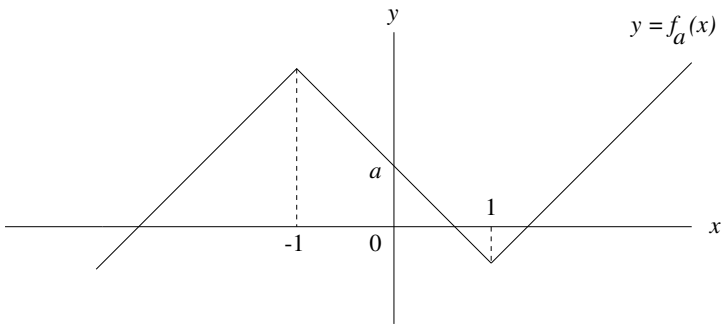


Figure 1

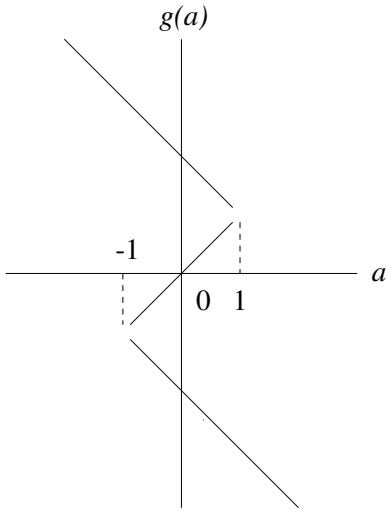


Figure 2

Note that other examples of a similar nature abound, and can be handled similarly; for example, the problem of finding, for a given real number x, an integer $n > x$.

2.3 Solutions for the abstract model

In the above three examples we have presented a number of single-valued functions $f : \mathbb{R}^n \to \mathbb{R}$ that we want to compute, and argued that:

(i) they are not continuous; and hence

(ii) they cannot be abstractly computed on the abstract data type containing $\mathbb{R}$;

(iii) however they can be computed in the concrete data type **CS**;

(iv) they are selection functions for many-valued functions on $\mathbb{R}$ that *are* continuous.

At the level of *concrete models* of computation, there is no real problem with the issues raised by these examples, since concrete models work only by computations on *representations* of the reals (say by Cauchy sequences), to be described in Sections 6 and 8.

The real problem arises with the construction of *abstract models* of computation on the reals which should model the phenomena illustrated by these examples, and should, moreover, correspond, in some sense, to the concrete models. Thus we have the question:

> *Can such continuous many-valued functions be computed on the abstract data type A containing $\mathbb{R}$ using new abstract models of computation? If they can, are the concrete and abstract models then equivalent?*

The rest of this paper deals with these issues. We answer the above question more generally, over many-sorted metric partial algebras A.

The solution presented in this paper is to extend the ***While**** programming language over A [Tucker and Zucker 2000] with a nondeterministic "countable choice" programming construct, so that in the rules of program term formation,

$$\mathsf{choose\ z} : b$$

is a new term of type nat, where z is a variable of type nat and b is a term of type bool. We will revisit the examples after giving the language definition in Section 4.

Alternatively (and equivalently), one could use other abstract models; *e.g.*, modify the $\mu\mathrm{PR}^*$ function schemes [Tucker and Zucker 2000, §9.1] by replacing the constructive least number (μ) operator, $f(x) \simeq \mu z \in \mathbb{N}[g(x,z) = \mathsf{tt}]$ (where g is boolean-valued) by a nondeterministic choice operator $f(x) \simeq \mathsf{choose}\ z \in \mathbb{N}[g(x,z) = \mathsf{tt}]$.

In [Brattka 1999] a more elaborate set of recursive schemes over many-sorted algebras, with many-valued operations, was presented.

3. TOPOLOGICAL PARTIAL ALGEBRAS AND CONTINUITY

We define some basic notions concerning topological and metric many-sorted partial algebras. Much of this information is in [Tucker and Zucker 2000], but we introduce here the concept of *partial algebra*, with examples which are important for later.

3.1 Basic algebraic definitions

A *signature* Σ (for a many-sorted partial algebra) is a pair consisting of (i) a finite set $\boldsymbol{Sort}(\Sigma)$ of *sorts*, and (ii) a finite set $\boldsymbol{Func}(\Sigma)$ of *typed function symbols* $F\colon u \to s$, where u is a *Σ-product type* $s_1 \times \cdots \times s_m$ $(m \geq 0)$, with $s_1, \ldots, s_m, s \in \boldsymbol{Sort}(\Sigma)$. (The case $m = 0$ corresponds to *constant symbols*.) We write $u, v, \ldots$ for Σ-product types.

A partial *Σ-algebra* A has, for each sort s of Σ, a non-empty *carrier set* A_s of sort s, and for each Σ-function symbol $F : u \to s$, a partial function $F^A : A^u \dot{\to} A_s$, where we write $A^u =_{df} A_{s_1} \times \cdots \times A_{s_m}$ if $u = s_1 \times \cdots \times s_m$. (The notation $f : X \dot{\to} Y$ refers to a partial function from X to Y.) We also write $\Sigma(A)$ for the signature of A.

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 paper we deal mainly with partial algebras. *The default assumption is that "algebra" refers to partial algebra.* We will, nevertheless, for the sake of emphasis, often speak explicitly of "partial algebras".

Examples 3.1.1. (*a*) The algebra of *booleans* has the carrier $\mathbb{B} = \{\mathsf{tt}, \mathsf{ff}\}$ of sort bool. The signature $\Sigma(\mathcal{B})$ and algebra $\mathcal{B}$ respectively can be displayed as follows:

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

and

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

Note that the signature can essentially be inferred from the algebra; indeed from now on we will not define the signature where no confusion will arise. Further, for notational simplicity, we will not always distinguish between function names in the signature (true, etc.) and their intended interpretations ($\mathsf{true}^{\mathcal{B}} = \mathsf{tt}$, etc.)

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

algebra	$\mathcal{N}_0$
carriers	$\mathbb{N}$
functions	$0 : \to \mathbb{N}$,
	$\mathsf{S} : \mathbb{N} \to \mathbb{N}$
end	

(*c*) The ring $\mathcal{R}_0$ of reals has a carrier $\mathbb{R}$ of sort real:

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

(*d*) The field $\mathcal{R}_1$ of reals is formed by adding the multiplicative inverse to the ring $\mathcal{R}_0$:

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

where

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

This is an example of a partial algebra. Other examples will be given later.

Throughout this work we make the following assumption about the signatures Σ.

ASSUMPTION 3.1.2 (INSTANTIATION ASSUMPTION). *For every sort s of Σ, there is a closed term of that sort, called the default term $\boldsymbol{\delta}^s$ of that sort.*

This guarantees the presence of *default values* $\boldsymbol{\delta}^s_A$ in a Σ-algebra A at all sorts s, and *default tuples* $\boldsymbol{\delta}^u_A$ at all product types u.

3.2 Adding booleans: Standard signatures and algebras

The algebra $\mathcal{B}$ of booleans (Example 3.1.1(a)) plays an essential role in computation. This motivates the following definition.

Definition 3.2.1 *(Standard signature).* A signature Σ is *standard* if (i) it is an expansion[1] of $\Sigma(\mathcal{B})$, and (ii) the function symbols of Σ include a *conditional*

$$\mathsf{if}_s : \mathsf{bool} \times s^2 \to s$$

for all sorts s of Σ other than bool.

For a standard Σ, a Σ-sort s is called an *equality sort* if Σ includes an *equality operator*

$$\mathsf{eq}_s : s^2 \to \mathsf{bool}.$$

Definition 3.2.2 *(Standard algebra).* Given a standard signature Σ, a Σ-algebra A is a *standard* if (i) it is an expansion of $\mathcal{B}$, (ii) the conditional operator on each sort s has its standard interpretation in A; *i.e.*, for $b \in \mathbb{B}$ and $x, y \in A_s$,

$$\mathsf{if}^A_s(b, x, y) \;=\; \begin{cases} x & \text{if } \; b = \mathtt{tt} \\ y & \text{if } \; b = \mathtt{ff}; \end{cases}$$

and (iii) the 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 either returns $\mathtt{tt}$ or diverges; and if they are not identical, then it either returns $\mathtt{ff}$ or diverges.

Remarks 3.2.3. (a) In practice, part (iii) of the above definition occurs as one of three cases. First, the case

$$\mathsf{eq}^A_s(x, y) \;=\; \begin{cases} \mathtt{tt} & \text{if } \; x = y \\ \mathtt{ff} & \text{otherwise,} \end{cases}$$

i.e., total equality, represents the situation where equality is "decidable" or "computable" at sort s, for example, when $s = \mathsf{nat}$. Second, the case

$$\mathsf{eq}^A_s(x, y) \;=\; \begin{cases} \mathtt{tt} & \text{if } \; x = y \\ \uparrow & \text{otherwise} \end{cases}$$

represents typically the situation where equality is "semidecidable". An example is given by the initial *term algebra* of an r.e. equational theory. Third, the case

$$\mathsf{eq}^A_s(x, y) \;=\; \begin{cases} \uparrow & \text{if } \; x = y \\ \mathtt{ff} & \text{otherwise,} \end{cases}$$

[1] Expansions of signatures and algebras are defined in [Tucker and Zucker 2000, Def. 2.6].

represents typically the situation where equality is "co-semidecidable". Examples are given by the data types of *streams* and *reals*; *cf.* the discussion in 2.1(c) and Example 3.2.4(c).

(b) Any many-sorted signature Σ can be *standardised* to a signature $\Sigma^{\mathcal{B}}$ by adjoining the sort bool together with the standard boolean operations; and, correspondingly, any algebra A can be standardised to an algebra $A^{\mathcal{B}}$ by adjoining the algebra $\mathcal{B}$ as well as the conditional and equality operators.

Example 3.2.4 *(Standard algebras).*
(a) The simplest standard algebra is the algebra $\mathcal{B}$ of the booleans (Example 3.1.1(a)).

(b) A *standard algebra of naturals* $\mathcal{N}$ is formed by standardising the algebra $\mathcal{N}_0$ (Example 3.1.1(b)), with (total) equality and order operations on $\mathbb{N}$:

algebra	$\mathcal{N}$
import	$\mathcal{N}_0$, $\mathcal{B}$
functions	$\mathsf{if}^{\mathcal{N}}_{\mathsf{nat}} : \mathbb{B} \times \mathbb{N}^2 \to \mathbb{N}$,
	$\mathsf{eq}^{\mathcal{N}}_{\mathsf{nat}}$, $\mathsf{ls}^{\mathcal{N}}_{\mathsf{nat}} : \mathbb{N}^2 \to \mathbb{B}$
end	

(c) *A standard partial algebra* $\mathcal{R}_p$ *on the reals* is formed similarly by standardising the field $\mathcal{R}_1$ (Example 3.1.1(d)), with partial equality and order operations on $\mathbb{R}$:

algebra	$\mathcal{R}$
import	$\mathcal{R}_1$, $\mathcal{B}$
functions	$\mathsf{if}^{\mathcal{R}}_{\mathsf{real}} : \mathbb{B} \times \mathbb{R}^2 \dot{\to} \mathbb{R}$,
	$\mathsf{eq}^{\mathcal{R}}_{\mathsf{real}}$, $\mathsf{ls}^{\mathcal{R}}_{\mathsf{real}} : \mathbb{R}^2 \dot{\to} \mathbb{B}$
end	

where

$$\mathsf{eq}^{\mathcal{R}}_{\mathsf{real}}(x,y) \;=\; \begin{cases} \uparrow & \text{if } x = y \\ \mathsf{ff} & \text{if } x \neq y \end{cases} \qquad \text{and} \qquad \mathsf{ls}^{\mathcal{R}}_{\mathsf{real}}(x,y) \;=\; \begin{cases} \mathsf{tt} & \text{if } x < y \\ \mathsf{ff} & \text{if } x > y \\ \uparrow & \text{if } x = y. \end{cases}$$

Discussion 3.2.5 *(Semicomputability and co-semicomputability).* The significance of the partial equality and order operations in Example (c) above, in connection with computability and continuity, has been touched on in §2.1(c). The *continuity* of partial functions will be discussed in §3.5 (and see in particular Example 3.5.4(b)). Regarding *computability*, these definitions are intended to capture the intuition of the "*semicomputability*" of order and "*co-semicomputability*" of equality on (a concrete model of) the reals. For given two reals x and y, represented (say) by their infinite decimal expansions, suppose their decimal digits are being read systematically, the n-th digit of both at step n. Then if $x \neq y$ or $x < y$, this will become apparent after finitely many steps, but no finite number of steps can confirm that $x = y$.

Throughout this paper, we will assume the following.

ASSUMPTION 3.2.6 (STANDARDNESS ASSUMPTION). *The signature Σ and Σ-algebra A are standard.*

3.3 Adding counters: N-standard signatures and algebras

The standard algebra $\mathcal{N}$ of naturals (Example 3.2.4(b)) plays, like $\mathcal{B}$, an essential role in computation. This motivates the following definitions.

Definition 3.3.1 *(N-standard signature).* A signature is *N-standard* if (i) it is standard, and (ii) it is an expansion of $\Sigma(\mathcal{N})$.

Definition 3.3.2 *(N-standard algebra).* Given an N-standard signature Σ, a corresponding Σ-algebra A is *N-standard* if it is an expansion of $\mathcal{N}$.

Note that any standard signature Σ can be N-*standardised* to a signature Σ^N by adjoining the sort nat and the operations 0, S, $\mathsf{eq}_{\mathsf{nat}}$, $\mathsf{ls}_{\mathsf{nat}}$ and $\mathsf{if}_{\mathsf{nat}}$. Correspondingly, any standard Σ-algebra A can be N-*standardised* to an algebra A^N by adjoining the carrier $\mathbb{N}$ together with the corresponding standard functions.

Examples 3.3.3 *(N-standard algebras).* (a) The simplest N-standard algebra is the algebra $\mathcal{N}$ (Example 3.2.4(b)).

(b) We can N-standardise the algebra $\mathcal{R}_p$ (Example 3.2.4(c)) to form the algebra $\mathcal{R}_p^N$.

3.4 Adding arrays: Algebras A^* of signature Σ^*

A standard signature Σ, and standard Σ-algebra A, can be expanded in two stages:

(1°) N-standardise these to form Σ^N and A^N, as in §3.3.

(2°) Define, for each sort s of Σ, the carrier A_s^* to be the set of *finite sequences* or *arrays* 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. Details are given in [Tucker and Zucker 2000, §2.7] and (an equivalent but simpler version) in [Tucker and Zucker 1999, §2.4].

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 algebras in general (unlike the case with $\mathbb{N}$).

3.5 Topological partial algebras

We now add topologies to our partial algebras, with the requirement of continuity for the basic partial functions.

Definition 3.5.1. Given two topological spaces X and Y, a partial function $f : X \rightharpoonup Y$ is *continuous* iff for every open $V \subseteq Y$, $f^{-1}[V]$ is open in X, where

$$f^{-1}[V] =_{df} \{ x \in X \mid x \in \mathbf{dom}(f) \text{ and } f(x) \in Y \}.$$

Remark 3.5.2. For later use, we recast this definition in the language of metric spaces. Given two metric spaces X and Y, a partial function $f : X \rightharpoonup Y$ is

continuous iff

$$\forall a \in \boldsymbol{dom}(f)\, \forall \epsilon > 0\, \exists \delta > 0\, \forall x \in \mathbf{B}(a,\delta)\, \big(x \in \boldsymbol{dom}(f) \wedge f(x) \in \mathbf{B}(f(a),\epsilon)\big).$$

Definition 3.5.3. (*a*) A *topological partial Σ-algebra* is a partial Σ-algebra with topologies on the carriers such that each of the basic Σ-functions is continuous.

(*b*) An (N-)*standard topological partial algebra* is a topological partial algebra which is also (N-)standard, such that the carriers $\mathbb{B}$ (and $\mathbb{N}$) have the discrete topology.

Examples 3.5.4. (*a*) *Discrete algebras:* The standard algebras $\mathcal{B}$ and $\mathcal{N}$ of booleans and naturals respectively (§§3.1, 3.3) are topological (total) algebras under the discrete topology. All functions on them are trivially continuous, since the carriers are discrete.

(*b*) The *partial real algebra* $\mathcal{R}_p$ (Example 3.2.4(*c*)) and its N-standardised version $\mathcal{R}_p^N$ (Example 3.3.3(*b*)) can be construed as topological algebras, where $\mathbb{R}$ has its usual topology, and $\mathbb{B}$ and $\mathbb{N}$ the discrete topology. Note that the partial operations $\mathsf{eq}^{\mathcal{R}}_{\mathsf{real}}$ and $\mathsf{ls}^{\mathcal{R}}_{\mathsf{real}}$ are continuous. (Recall the discussion in (1.1(*c*)).)

(*c*) *Partial interval algebras* on the closed interval $[0,1]$ have the form

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

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	$\mathsf{i}_I : I \to \mathbb{R}$,
	$\ldots$
end	

(*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	$\mathsf{if}^{\mathcal{R}}_{\mathsf{real}} : \mathbb{B} \times \mathbb{R}^2 \to \mathbb{R}$,
	$\mathsf{div}^{\mathcal{R}}_{\mathsf{nat}} : \mathbb{R} \times \mathbb{N} \to \mathbb{R}$,
end	

Here $\mathcal{R}_0$ is the ring of reals (example 3.1.1(*c*)), $\mathcal{N}$ is the standard algebra of naturals (3.2.4(*b*)), and $\mathsf{div}_{\mathsf{nat}}$ is division of reals by naturals (total and continuous! — just let $\mathsf{div}_{\mathsf{nat}}(x^{\mathsf{real}}, 0^{\mathsf{nat}}) =_{df} 0^{\mathsf{real}}$).

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 $\mathsf{ls}_{\mathsf{real}}$ of $\mathcal{R}_p$).

Definition 3.5.5 *(Extensions of topology to A^N and A^*).* The various algebraic expansions of A detailed in §§3.3/3.4 induce corresponding topological expansions.
(a) The topological N-standardisation A^N, of signature Σ^N, is constructed from A by giving the new carrier $\mathbb{N}$ the discrete topology.
(b) The topological array algebra A^*, of signature Σ^*, is constructed from A^N by giving A_s^* the disjoint union topology of the sets $(A_s)^n$ of arrays of length n, for all $n \geq 0$, where each set $(A_s)^n$ is given the product topology of the sets A_s.

It can be seen that this is the topology on A^* generated by the new functions, *i.e.*, the weakest topology which makes them continuous. It can also be described as follows. The *basic open sets* in A_s^* have the form

$$\{\, a^* \in A_s^* \mid \mathsf{Lgth}(a^*) = n \ \text{ and } \ a^*[i_1] \in U_1,\ \ldots,\ a^*[i_k] \in U_k \,\}$$

for some $n, k, i_1, \ldots, i_k$, where $0 \leq k < n$ and $0 \leq i_1 < \cdots < i_k < n$, and for some open sets $U_1, \ldots, U_k \subseteq A_s$.

3.6 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	$F_1^A : A^{u_1} \to A_{s_1}$,
	$\ldots$
	$F_k^A : A^{u_k} \to A_{s_k}$,
	$\mathsf{d}_1^A : A_1^2 \to \mathbb{R}$,
	$\ldots$
	$\mathsf{d}_r^A : A_r^2 \to \mathbb{R}$
end	

where $\mathcal{B}$ and $\mathcal{R}_p$ are respectively the algebras of booleans and reals (Examples 3.1.1(a), 3.2.4(c)), the carriers $A_1, \ldots, A_r$ are metric spaces with metrics $\mathsf{d}_1^A, \ldots$ $\ldots, \mathsf{d}_r^A$ respectively, $F_1, \ldots, F_k$ are the Σ-function symbols other than $\mathsf{d}_1, \ldots, \mathsf{d}_r$, and the (partial) functions F_i^A are all continuous with respect to these metrics (*cf.* Definition 3.5.1).

Note that the carrier $\mathbb{B}$ (as well as $\mathbb{N}$, if present) has the *discrete metric*, defined by

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

which induces the discrete topology.

We will often speak of a "metric algebra A", without stating the metric explicitly.

Example 3.6.1. Clearly, metric partial algebras can be viewed as special cases of topological partial algebras. Thus the partial and total real algebras $\mathcal{R}_p$, $\mathcal{R}_p^N$ and $\mathcal{R}_t^N$ (Examples 3.5.4) can be recast as metric algebras in an obvious way.

Remark 3.6.2 *(Extension of metric to A^*).* A metric algebra A can be expanded to a metric algebra A^* of arrays over A. Namely, given a metric d_s on A_s, we define a (bounded) metric $\mathsf{d}^*{}_s$ on A_s^* as follows: for $a^* = (a_1, \ldots, a_k)$, $b^* = (b_1, \ldots, b_l) \in A_s^*$:

$$\mathsf{d}^*{}_s(a^*, b^*) \;=\; \begin{cases} 1 & \text{if } \; k \neq l \\ \min\big(1,\; \max_{i=0}^{k-1} \mathsf{d}_s(a^*[i], b^*[i])\big) & \text{otherwise.} \end{cases}$$

This gives the topology on A^* induced by the topology on A (Definition 3.5.5) [Engelking 1989].

Remark 3.6.3 *(Product metric on A).* If A is a Σ-metric algebra, then for each Σ-product sort $u = s_1 \times \cdots \times s_m$, we can define a metric d_u on A^u by

$$d_u((x_1, \ldots, x_m), (y_1, \ldots, y_m)) \;=\; \max_{i=1}^{m}\big(d_{s_i}(x_i, y_i)\big)$$

or more generally, by the ℓ_p metric

$$d_u((x_1, \ldots, x_m), (y_1, \ldots, y_m)) \;=\; \big(\sum_{i=1}^{m}(d_{s_i}(x_i, y_i))^p\big)^{1/p} \qquad (1 \leq p \leq \infty)$$

where $p = \infty$ corresponds to the "max" metric. This induces the product topology on A^u.

Remark 3.6.4 *(W-continuity).* An alternative notion of continuity of partial functions, used by Weihrauch and others [Weihrauch 2000; Brattka 1996], is discussed in Appendix A.

4. '*While*' PROGRAMMING WITH COUNTABLE CHOICE

The programming language $\boldsymbol{WhileCC} = \boldsymbol{WhileCC}(\Sigma)$ is an extension of $\boldsymbol{While}(\Sigma)$ [Tucker and Zucker 2000, §3] with an extra 'choose' rule of term formation. We give the complete definition of its syntax and semantics, using the *algebraic operational semantics* of [Tucker and Zucker 2000].

Assume Σ is an N-standard signature, and A is an N-standard Σ-algebra.

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

We define four syntactic classes: *variables*, *terms*, *statements* and *procedures*.

(*a*) $\boldsymbol{Var} = \boldsymbol{Var}(\Sigma)$ 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: $\mathtt{a}^s, \mathtt{b}^s, \ldots, \mathtt{x}^s, \mathtt{y}^s, \ldots$.

(*b*) $\boldsymbol{PTerm} = \boldsymbol{PTerm}(\Sigma)$ is the class of Σ-*program terms* $t, \ldots,$ and for each Σ-sort s, $\boldsymbol{PTerm}_s$ is the class of program terms of sort s. These are generated by the rules

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

where $s, s_1, \ldots, s_n$ are Σ-sorts, $F : s_1 \times \cdots \times s_n \to s$ is a Σ-function symbol, $t_i \in \boldsymbol{PTerm}_{s_i}$ for $i = 1, \ldots, n$ $(n \geq 0)$, and b is a boolean term, *i.e.*, a term of sort bool.

The 'choose' term has sort nat. Think of 'choose' as a generalisation of the *constructive least number operator* $\mathsf{least}\ \mathsf{z} : b$ which has the value k in case $b[\mathsf{z}/k]$

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 '$\mathsf{choose}\ \mathsf{z} : b$' selects *some* value k such that $b[\mathsf{z}/k]$ is true, if any such k exists (and is undefined otherwise). In our abstract semantics, we will give the meaning as 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 construction, but rather as a new atomic program statement: '$\mathsf{choose}\ \mathsf{z} : b$'. We prefer the present treatment, as it leads to the construction of *many-valued term semantics* (as we will see), which is interesting in itself, and which we would have to deal with anyway if we were to extend our syntax to include (many-valued) function procedure calls in our term construction.

We write $t : s$ to indicate that $t \in \boldsymbol{PTerm}_s$, and for $u = s_1 \times \cdots \times s_m$, 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 $b, \ldots$ for boolean terms.

(*c*) $\boldsymbol{AtSt} = \boldsymbol{AtSt}(\Sigma)$ is the class of *atomic statements* $S_{\mathsf{at}}, \ldots$ defined by

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

where 'div' stands for "divergence" (non-termination), and $\mathsf{x} := t$ is a *concurrent assignment*, where for some product type u, $t : u$ and x is a u-tuple of *distinct* variables.

(*d*) $\boldsymbol{Stmt} = \boldsymbol{Stmt}(\Sigma)$ is the class of statements $S, \ldots$, generated by the rules

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

(*e*) $\boldsymbol{Proc} = \boldsymbol{Proc}(\Sigma)$ is the class of function procedures $P, Q, \ldots$. These have the form

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

where a, b and c are lists of *input variables*, *output variables* and *auxiliary* (or *local) variables* respectively, and S is the *body*. Further, we stipulate:

—a, b and c each consist of distinct variables, and they are pairwise disjoint,

—all variables occurring in S must be among a, b or c,

—the *input variables* a must not occur on the lhs of assignments in S,

—*initialisation condition*: S has the form $S_{init}; S'$, where S_{init} is a concurrent assignment which initialises all the *output* and *auxiliary variables*, *i.e.*, assigns to each variable in b and c the default term (3.1.2) of the same sort.

If $\mathsf{a} : u$ and $\mathsf{b} : v$, then P is said to have *type* $u \to v$, written $P : u \to v$. Its *input type* is u and its *output type* is v.

4.2 Algebraic operational semantics of *WhileCC*

We interpret programs as countably-many-valued state transformations, and function procedures as countably-many-valued functions on A. Our approach follows the *algebraic operational semantics* of [Tucker and Zucker 2000, §3.4]. First we need some notation.

Notation 4.2.1.
(*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 $f : X \rightrightarrows Y$ for $f : X \to \mathcal{P}_\omega(Y)$.
(*e*) We write $f : X \rightrightarrows^+ Y$ for $f : X \to \mathcal{P}^+_\omega(Y)$.

We will interpret a ***WhileCC*** procedure $P : u \to s$ as a countably-many-valued function P^A from A^u to ${A_s}^\uparrow$, *i.e.*, as a function

$$P^A : A^u \to \mathcal{P}^+_\omega({A_s}^\uparrow)$$

or, in the above notation:

$$P^A : A^u \rightrightarrows^+ {A_s}^\uparrow.$$

Remark 4.2.2 *(Significance of '$\uparrow$').* Notice that an output of, say, $\{2, 5, \uparrow\}$ is different from $\{2, 5\}$, since the former indicates the possibility of divergence. So a semantic function will have, for inputs not in its domain, '$\uparrow$' as a possible output value.

Definition 4.2.3 *(States).* (*a*) For each Σ-algebra A, a *state* on A is a family $\langle \sigma_s \mid s \in \boldsymbol{Sort}(\Sigma)\rangle$ of functions

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

Let $\boldsymbol{State}(A)$ be the set of states on A, with elements $\sigma, \ldots$.

(*b*) 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 $i = 1, \ldots, n$.

We give a brief overview of *algebraic operational semantics*. This was used in [Tucker and Zucker 1988] for deterministic imperative languages with 'while' and recursion (see [Tucker and Zucker 2000] for the case of $\boldsymbol{While}(\Sigma)$), but it can be applied to a wide variety of imperative languages. It has also been used to analyse compiler correctness [Stephenson 1996]. It can also be adapted, as we will see, to a nondeterministic language such as ***WhileCC****.

Assume (*i*) we have a meaning function for atomic statements

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

and (*ii*) we have defined a pair of functions

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

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 (or, in the present nondeterministic context, a finite set of statements) which gives the rest of the execution in state* σ.

From these we define the *computation step* function

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

by

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

from which, in turn, we can define (for the deterministic language of [Tucker and Zucker 2000]) a *computation sequence* or (for the present language) a *computation tree*. The aim is to define a *computation tree stage* function

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

where $\boldsymbol{CompTreeStage}^A(S,\sigma,n)$ represents the first n stages of $\boldsymbol{CompTree}^A(S,\sigma)$. Here $(\boldsymbol{State}(A)^\uparrow)^{<\omega}$ denotes the set of finite sequences from $\boldsymbol{State}(A)^\uparrow$, interpreted as finite initial segments of the paths through the computation tree. From this are defined the semantics of statements and procedures.

Remark 4.2.4. The intuition behind these semantics is that for any input $x \in A^u$, $P^A(x)$ is the set of all possible outcomes (including divergence), for all possible implementations of the 'choose' construct, *including non-constructive implementations*! So if (for a given input x) the only infinite paths through the semantic computation tree are non-constructive, then $P^A(x)$ will still include '$\uparrow$'. This is discussed further in §4.4(b).

We turn to the details of these definitions.

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

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

The definition is by structural induction on t:

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

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

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

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

$$\boldsymbol{First}\colon\ \boldsymbol{Stmt} \;\to\; \boldsymbol{AtSt}$$

is defined exactly as in [Tucker and Zucker 2000, §3.5], namely:

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

The operation

$$\boldsymbol{Rest}^A\colon\ \boldsymbol{Stmt} \times \boldsymbol{State}(A) \;\rightrightarrows^+\; \boldsymbol{Stmt},$$

is defined as follows (*cf.* [Tucker and Zucker 2000, §3.5]):

Case 1. S is atomic. Then $\boldsymbol{Rest}^A(S,\sigma) \;=\; \{\,\mathsf{skip}\,\}$.

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 $\boldsymbol{Rest}^A(S,\sigma) \;=$

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

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

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

Note that more than one condition may hold.

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

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

Note again that more than one condition may hold.

(*d*) Computation step. From ***First*** we can define the computation step function

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

which is like the one-step computation function $\boldsymbol{Comp}_1^A$ of [Tucker and Zucker 2000, §3.4], 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 in [Tucker and Zucker 2000], 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 $\boldsymbol{Comp}^A$ of [Tucker and Zucker 2000, §3.4] is replaced by a *computation tree stage* function

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

where $\boldsymbol{CompTreeStage}^A(S,\sigma,n)$ represents the first n stages of $\boldsymbol{CompTree}^A(S,\sigma)$. This is defined (like $\boldsymbol{Comp}^A$) by a simple recursion ("tail recursion") on n:

Basis: $\boldsymbol{CompTreeStage}^A(S,\sigma,0) = \{\sigma\}$, *i.e.*, just the root labelled by σ.

Induction step: $\boldsymbol{CompTreeStage}^A(S,\sigma,n)$ is formed by attaching to the root $\{\sigma\}$ the following:

(i) for S atomic: the leaf $\{\sigma'\}$, for each $\sigma' \in \langle\!|S|\!\rangle^A\sigma$ (where σ' may be a state or $\uparrow$);

(ii) for S not atomic: the subtree $\boldsymbol{CompTreeStage}^A(S',\sigma',n-1)$, for each $\sigma' \in \boldsymbol{CompStep}^A(S,\sigma)$ $(\sigma' \neq \uparrow)$ and $S' \in \boldsymbol{Rest}^A(S,\sigma)$, as well as the leaf $\{\uparrow\}$ if '$\uparrow$' $\in \boldsymbol{CompStep}^A(S,\sigma)$.

Then $\boldsymbol{CompTree}^A(S,\sigma)$ is defined as the "limit" over n of $\boldsymbol{CompTreeStage}^A(S,\sigma,n)$.

Note that only the leaves of $\boldsymbol{CompTree}^A(S,\sigma)$ may contain '$\uparrow$' ("local divergence").

(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 '$\uparrow$' at all leaves in $\boldsymbol{CompTree}^A(S,\sigma)$, together with '$\uparrow$' 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) '$\uparrow$' 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 '$\uparrow$' leaf.

(g) Semantics of procedures. Finally, if

$$P \equiv \mathsf{func\ in\ a\ out\ b\ aux\ c\ begin}\ S\ \mathsf{end} \tag{4}$$

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

$$P^A : A^u \rightrightarrows^+ A^{v\uparrow}$$

defined as follows (*cf.* [Tucker and Zucker 2000, §3.6]). For $x \in A^u$,

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

where σ is any state on A such that $\sigma[\mathtt{a}] = x$. (From the initialisation condition (§4.1(e)) it follows by a "functionality lemma" [Tucker and Zucker 2000, 3.6.1] that P^A is well defined.)

Definition 4.2.5. A ***WhileCC*** procedure $P : u \to v$ is *deterministic* on A if for all $x \in A^u$, $P^A(x)$ is a singleton.

Remark 4.2.6 *(Two concepts of deterministic computation).* One can distinguish between two notions of deterministic computation: (i) *strong deterministic computation*, the common concept, in which each step of the computation is determinate; and (ii) *weak deterministic computation*, in which the output (or divergence) is uniquely determined by (*i.e.*, a unique function of) the input, but the steps in the computation are *not* necessarily determinate. A good example of (ii) is the Gaussian elimination algorithm (examples 2.2.1 and 5.2.1) which, although defining a unique function (the inverse of a matrix), incorporates the (nondeterministic!) pivot function as a subroutine. In Definition 4.2.5 and elsewhere in this paper, we are concerned with the weak sense of deterministic computation.

Definition 4.2.7.

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

(b) A partial function $f : A^u \rightharpoonup A_s$ is ***WhileCC*** *computable* on A if there is a deterministic ***WhileCC*** procedure $P : u \to s$ such that for all $x \in A^u$,
(i) $f(x) \downarrow y \implies P^A(x) = \{y\}$, and
(ii) $f(x)\uparrow \implies P^A(x) = \{\uparrow\}$,

Remark 4.2.8 *(Many-valued algebras).* As we have seen, 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 [Brattka 1996; 1999], the algebraic operational semantics could handle this just as easily, by adapting the clause for the basic Σ-function f in part (a) ("Semantics of program terms") of the semantic definition above.

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

In [Tucker and Zucker 1999; 2000] we worked with the language $\boldsymbol{While}^*(\Sigma)$ (rather than $\boldsymbol{While}(\Sigma)$), formed by augmenting ***While*** with auxiliary array and nat variables [Tucker and Zucker 2000, §3.13]. 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 [Tucker and Zucker 2000, §8].

Here, similarly, we will work with the language $\boldsymbol{WhileCC}^* = \boldsymbol{WhileCC}^*(\Sigma)$, which can be viewed similarly as $\boldsymbol{WhileCC}(\Sigma)$ augmented by auxiliary array and nat 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 or sort nat.)

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

4.4 Some computability issues in the semantics of $\boldsymbol{WhileCC}^*$ procedures

Some interesting issues in the semantics of $\boldsymbol{WhileCC}^*$ arise already in the case of computation over the algebra $\mathcal{N}$ of naturals (Example 3.2.4(b)).

(a) Eliminating 'choose' from deterministic $\boldsymbol{WhileCC}^*$ on total algebras

The 'choose' operator can be eliminated from deterministic $\boldsymbol{WhileCC}^*$ procedures (*cf.* Definition 4.2.5 and Remark 4.2.6) over total algebras.

THEOREM 4.4.1. *For any total Σ-algebra A and $f\colon A^u \dot{\to} A_s$,*

f is $\boldsymbol{WhileCC}^*$ computable over A $\iff$ f is $\boldsymbol{While}^*$ computable over A.

PROOF. ($\Rightarrow$) Let P be a deterministic $\boldsymbol{WhileCC}^*$ procedure over A which computes f. Since A is total, evaluation of any boolean term b over A (relative to a state) converges to tt or ff in A. Further, since P is deterministic, its output for a given input is independent of the implementation. Hence every 'choose' term in P of the form choose z : b[z] can be replaced by a 'while' loop which tests $b[0]$, $b[1]$, $b[2]$, ... in turn, *i.e.*, finds the *least* k for which $b[k]$ is true, if it exists, and diverges otherwise. □

Applying this to the total algebra $\mathcal{N}$, and recalling that $\boldsymbol{While}^*$ computability over $\mathcal{N}$ is equivalent to *partial recursiveness* (*i.e.*, classical computability) over $\mathbb{N}$ [Tucker and Zucker 2000], we have:

COROLLARY 4.4.2. *For any $f\colon \mathbb{N}^m \dot{\to} \mathbb{N}$,*

f is $\boldsymbol{WhileCC}^*$ computable over $\mathcal{N}$ $\iff$ f is partial recursive over $\mathbb{N}$.

(b) Recursive and non-recursive implementations

The semantics P^A of a procedure P (§4.2) is given, for an input x, by *all paths* of the computation tree $T = \boldsymbol{CompTree}^A(S, \sigma)$ (where S is the body of P) representing *all possible computation sequences* for S starting at state σ, where σ[a] $= x$, *i.e.*, all possible implementations of instances of the 'choose' construct occurring in the execution of S starting at σ. This leads to interesting computation-theoretic issues even in the simple case that $A = \mathcal{N}$, where we can assume that T is coded as a subset of $\mathbb{N}$ in a standard way. Now any path of T ending in a leaf is finite, and therefore (trivially) recursive. An infinite path or computation sequence (leading to divergence), however, may or may not be recursive. (See Remark 4.2.4.)

THEOREM 4.4.3. *There is a $\boldsymbol{WhileCC}^*(\mathcal{N})$ procedure P such that its computation tree has an infinite path, but no recursive infinite path.*

The construction of P is based on the construction of a recursive tree with an infinite path, but no recursive infinite path [Odifreddi 1999, V.5.25]. Details are given in Appendix B.

For this procedure P, $\uparrow \in P^A()$, *i.e.*, divergence is possible. However, if we were to restrict computation sequences to be recursive, then divergence would not be a

possible outcome for $P^A()$. The semantics, as we give it (*i.e.*, all possible computation sequences included, whether recursive or not) is simpler than this alternative. In any case, as we will see, this choice will not affect continuity considerations (*cf.* Lemmas 6.1.7 and 6.2.1).

4.5 Approximable $\boldsymbol{WhileCC}^*$ computability

The basic notion of computability that we will be using in working with metric algebras is not so much computability, as rather *computable approximability on metric algebras*, as discussed in [Tucker and Zucker 1999, §9]. We have to adapt the definition given there 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 $\boldsymbol{WhileCC}^*(\Sigma^N)$ procedure. Put

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

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

Definition 4.5.1 (*$\boldsymbol{WhileCC}^*$ approximability to a single-valued function).* Let $f : A^u \rightharpoonup A_s$ be a single-valued partial function on A.

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

$$x \in \boldsymbol{dom}(f) \implies \uparrow \notin P_n^A(x) \subseteq \mathbf{B}(f(x), 2^{-n}). \tag{5}$$

(*b*) f is *strictly $\boldsymbol{WhileCC}^*$ approximable* by P on A if in addition to (5),

$$x \notin \boldsymbol{dom}(f) \implies P_n^A(x) = \{\uparrow\}. \tag{6}$$

Remarks 4.5.2.

(*a*) Clearly, *$\boldsymbol{WhileCC}^*$ computability* is a special case of *$\boldsymbol{WhileCC}^*$ approximability*.

(*b*) For total f, the concepts of $\boldsymbol{WhileCC}^*$ approximability and strict $\boldsymbol{WhileCC}^*$ approximability coincide.

(*c*) If a single-valued function f is strictly approximable by P, then (from (5) and (6)) for all $x \in A^u$ and all n:

$$f(x)\uparrow \iff \uparrow \in P_n^A(x) \iff P_n^A(x) = \{\uparrow\}.$$

Definition 4.5.3 (*$\boldsymbol{WhileCC}^*$ approximability to a many-valued function).* Let $f : A^u \rightrightarrows A_s$ be a countably-many-valued function on A.

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

$$\begin{array}{rcl} f(x) \neq \emptyset \implies \uparrow \notin P_n^A(x) & \subseteq & \bigcup_{y \in f(x)} \mathbf{B}(y, 2^{-n}) \\ \text{and} \quad f(x) & \subseteq & \bigcup_{y \in P_n^A(x)} \mathbf{B}(y, 2^{-n}). \end{array} \tag{7}$$

Note that (assuming $\uparrow \notin P_n^A(x)$) the r.h.s. of (7) implies

$$\mathsf{d}_{\mathsf{H}}(\overline{f(x)}, \overline{P_n^A(x)}) \leq 2^{-n}, \tag{8}$$

and is implied by

$$\mathsf{d_H}(\overline{f(x)},\ \overline{P_n^A(x)}) < 2^{-n}, \tag{9}$$

where $\overline{X}$ denotes the closure of X, and $\mathsf{d_H}$ is the *Hausdorff metric* on the set of closed, bounded non-empty subsets of A_s [Engelking 1989, 4.5.23]. (Actually, the Hausdorff metric applies only to the space of closed *bounded* subsets of a given metric space, so (8) and (9) should be taken as heuristic statements.)
In other words (assuming $f(x) \neq \emptyset$), for all $x \in A^u$ and all n, *each* output of $f(x)$ lies within 2^{-n} of some output of $P_n^A(x)$, and vice versa.

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

$$f(x) = \emptyset \implies P_n^A(x) = \{\uparrow\}.$$

Remark 4.5.4. (*Cf.* Remark 4.5.2(*c*).) If a many-valued function f is strictly approximable by P, then for all $x \in A^u$ and all n:

$$f(x) = \emptyset \iff \uparrow \in P_n^A(x) \iff P_n^A(x) = \{\uparrow\}.$$

5. EXAMPLES OF ***WhileCC**** EXACT AND APPROXIMATING COMPUTATIONS

5.1 Discussion: Use of 'choose' for searching and dovetailing

Following the examples in Section 2, the 'choose' construct was introduced to compute many-valued functions. Technically, this construct strengthens the power of the ***While*** language in performing searches. In a *partial algebra*, simple searches (*e.g.*, "find some x_k in an effectively enumerated set $X = \{x_0, x_1, x_2, \dots\}$ satisfying $b(x_k)$") will obviously fail in general if the search simply follows the given enumeration of X (*i.e.*, testing in turn whether $b(x_0)$, $b(x_1)$, $b(x_2)$, ...holds), since the computation of the boolean predicate $b(x)$ may not terminate for some x.

This problem is overcome, at the *concrete model* level, by the use of scheduling techniques such as *interleaving* or "*dovetailing*": at stage n, do n steps in testing whether $b(x_i)$ holds, for $i = 0, \dots, n$.

An important function of 'choose', which will recur in our examples, is to simulate such scheduling techniques at the *abstract model* level. This allows searches over any countable subset X of an algebra A that has a computable enumeration $\mathsf{enum}_X : \mathbb{N} \to X$, since we can search X in A by assignments such as

$$\mathtt{x} := \mathsf{enum}_X(\mathsf{choose}\ \mathtt{z} : b(\mathsf{enum}_X(\mathtt{z}))).$$

5.2 Examples

We now illustrate the use of the ***WhileCC**** language in topological partial algebras with examples, which involve computations which are either many-valued, or approximating, or both. The examples given in §2.2 to motivate many-valued abstract computation are a good place to start. They can be displayed in the table:

	Exact computation	Approximating computation
Single-valued	Gaussian elimination	e^x, $\sin(x)$, etc.
Many-valued	Approx. points in metric algebra	All simple roots of polynomial

Examples 5.2.1, 5.2.2 and 5.2.4 below are all based on the metric algebra derived from $\mathcal{R}_p^N$ (Example 3.3.3(*b*)).

Example 5.2.1 *(Gaussian elimination).* This is a single-valued exact computation. The algorithm can be found in any standard text of numerical computation, *e.g.*, [Heath 1997]. It is deterministic, but only in the weak sense (*cf.* Remark 4.2.6), since it contains, as an essential component, the computation of the *pivot* function (§2.2), which is many-valued, and can be formalised simply with the 'choose' construct:

```
func in x_1, ..., x_n : real
     out i : nat
     aux k : nat
begin
     i := choose k : (k = 1 and x_1 ≠ 0) or
                     (k = 2 and x_2 ≠ 0) or
                                ...
                     (k = n and x_n ≠ 0)
end.
```

Example 5.2.2 *(Approximations to e^x).* On the interval algebra $\mathcal{I}_p^N$ (Example 3.5.4(c)) we give a ***While*** procedure to approximate the function e^x on I:

```
func in  n : nat,     { degree of approximation }
         x : intvl    { 'intvl' is the sort of reals in the interval [0, 1] }
     out s : real     { partial sum of power series }
     aux y : real,    { current term of series }
         k : nat      { counter }
begin
     k := 0;
     y := 1;
     s := 1;
     while k < 2^(n+1) do
          k := k + 1;
          y := y × i_I(x)/i_N(k);        { y = x^k/k! }
          s := s + y                     { s = Σ_{i=0}^{k} x^i/i! }
     od
end
```

Here $\mathsf{i}_I : I \to \mathbb{R}$ is the embedding of I in $\mathbb{R}$, which is primitive in $\Sigma(\mathcal{I}_p^N)$, and $\mathsf{i}_N : \mathbb{N} \to \mathbb{R}$ is the embedding of $\mathbb{N}$ in $\mathbb{R}$, which is easily definable in ***While***$(\mathcal{R}_p^N)$.

Denoting the above function procedure by P, and $\mathcal{I}_p^N$ by A, we have the semantics

$$P_n^A : I \to \mathbb{R}$$

with

$$P_n^A(x) = \sum_{i=0}^{2^{n+1}} \frac{x^i}{i\,!}$$

and so for all $x \in I$,

$$\mathsf{d}(P_n^A(x),\, e^x) < 2^{-n},$$

i.e., e^x is ***While*** approximable on $\mathcal{I}_p^N$ by P.

This computation of e^x is single-valued, but approximating.

Example 5.2.3. *("Choosing" a member of an enumerated subspace close to an arbitrary element of a metric algebra.)* Given a metric algebra A with a countable dense subspace C, and an enumeration $\mathsf{enum}_C : \mathbb{N} \twoheadrightarrow C$ of C in the signature, we want to compute a function $f : A \times \mathbb{N} \to C$ such that

$$f(a, n) \;=\; \text{"some" } x \in C \text{ such that } \mathsf{d}(a, x) \;<\; 2^{-n}.$$

This is a generalisation of the problem of approximating reals by rationals (Example 2.2.3).

Here is a ***WhileCC**** procedure (in pseudo-code) for an exact computation of f. (Note that the real-valued function 2^{-n} is ***While*** computable on $\mathcal{R}_p^N$, and hence on A.)

```
func in   a : space,  n : nat
     out  x : space
     aux  k : nat
begin
     x := enum_C (choose k : d(a, enum_C(k)) < 2^-n)
end
```

This computation is many-valued, but exact.

Example 5.2.4 *(Finding simple roots of a polynomial).* We construct a ***WhileCC*** procedure to approximate "some" simple root of a polynomial $p(X)$ with real coefficients, using the method of bisection. By a *simple root of* $p(X)$ we mean a real root at which $p(X)$ changes sign. (See [Heath 1997]. In practice, a hybrid method is generally used, involving bisection, Newton's method, etc.)

Fundamental to the bisection method is the concept of a *bracket* for $p(X)$, which means an interval $[a, b]$ such that $p(a)$ and $p(b)$ have opposite signs. By *rational bracket*, we mean a bracket with rational endpoints. We note the following:

(1) Any bracket for p contains a root of p (by the Intermediate Value Theorem), in fact a simple root of p.

(2) Conversely, any simple root of p is contained in a rational bracket for p of arbitrarily small width.

(3) If x is a simple root of p, then any bracket for p of sufficiently small width which contains x, contains no other simple root of p.

(4) If $[a, b]$ is a bracket for p, then, putting $m = (a + b)/2$, exactly one of the following holds:
   (i) $p(m) = 0$; then m is a root of p (not necessarily simple);
   (ii) $p(m)$ has the same sign as $p(a)$; then $[m, b]$ is a bracket for p;
   (iii) $p(m)$ has the same sign as $p(b)$; then $[a, m]$ is a bracket for p.

It follows from the above that starting with any rational bracket J for p, we can, by repeated bisection, find a nested sequence of rational brackets

$$J = J_0,\ J_1,\ J_2, \ldots \qquad \text{where} \qquad \bigcap_{n=0}^{\infty} J_n = \{x\}$$

for some simple root x of p. Then, letting r_n be the left-hand endpoint of J_n, we have a fast Cauchy sequence $\langle r_n \rangle_n$ with limit x.

One complication with our algorithm is the occurrence of case (i) in (4) above, *i.e.*, the case that the midpoint m of the bracket is itself a root of p, since by the co-semicomputability of equality (Discussion 3.2.5) on $\mathbb{R}$ we can only verify when $f(m) \neq 0$, not when $f(m) = 0$. We therefore proceed as follows. By means of the 'choose' construct, we search in the middle third (say) of the bracket $[a, b]$ for a "division point", *i.e.*, a rational point d such that $f(d) \neq 0$, producing either $[a, d]$ or $[d, b]$ as a sub-bracket. (So we use a "trisection", instead of "bisection", method.)

This new bracket may not halve the width of $[a, b]$; in the worst case its width is $2/3(b-a)$. However a second iteration of this procedure leads to a bracket of width at most $(2/3)^2 < 1/2$ the width of $[a, b]$, and so $2n$ iterations lead to a bracket of width less than $2^{-n}(b-a)$.

For convenience, we will use the following two conservative extensions to our "official" programming notation:

(a) Simultaneously choosing two naturals with a single condition:

$$\mathtt{k}_1, \mathtt{k}_2 \;:=\; \mathsf{choose}\ \mathtt{z}_1, \mathtt{z}_2 : b[\mathtt{z}_1, \mathtt{z}_2]$$

which is easily expressible in ***WhileCC*** by the use of a primitive recursive pairing function pair on $\mathbb{N}$ and its inverses $\mathsf{proj}_1, \mathsf{proj}_2$:

$$\begin{aligned} \mathtt{k} \;&:=\; \mathsf{choose}\ \mathtt{z} : b[\mathsf{proj}_1(\mathtt{z}), \mathsf{proj}_2(\mathtt{z})]; \\ \mathtt{k}_1, \mathtt{k}_2 \;&:=\; \mathsf{proj}_1(\mathtt{k}), \mathsf{proj}_2(\mathtt{k}). \end{aligned}$$

(b) Choosing a rational (of type real) satisfying a boolean condition:

$$\mathtt{q} \;:=\; \mathsf{choose}\ \mathtt{r}^{\mathsf{real}} : \big(\text{“}\mathtt{r} \text{ is rational”}\ \mathsf{and}\ b[\mathtt{r}]\big).$$

Let $\mathsf{rat} : \mathbb{N} \to \mathbb{R}$ be a ***While***-computable enumeration of the rationals in $\mathbb{R}$. Then this can be interpreted as:

$$\mathtt{q} := \mathsf{rat}\big(\mathsf{choose}\ \mathtt{k} : b[\mathsf{rat}(\mathtt{k})]\big).$$

Finally, a polynomial $p(X)$ over $\mathbb{R}$ will be represented by an element p^* of $\mathbb{R}^*$:

$$p^* \;=\; (a_0, \ldots, a_{n-1}) \;=\; \sum_{i=0}^{n-1} a_i X^{n-i}.$$

Its evaluation at a point c, denoted by $p^*(c)$, is easily seen to be ***While***$(\mathcal{R})$ computable in p^* and c.

Hence we can give a ***WhileCC**** procedure for approximably computing some simple root of an input polynomial, in the signature of $\mathcal{R}_p$ (see Figure 3).
For input natural n and polynomial p, the output is within 2^{-n} of some simple root of p. Further, for *any* simple root e of p, there is *some* implementation of the 'choose' operator which will give an output within 2^{-n} of e. Finally, the computation will diverge if, and only if, p has no simple roots.

This computation is both many-valued and approximating.

```
func in    n : nat,        { degree of approximation }
           p* : real*      { input polynomial, given by list of coefficients }
     out   x : real        { approximation to root }
     aux a, b : real,      { endpoints of bracket }
           d : real,       { division point of bracket }
           k : nat         { counter }
begin
     k := 0;
     a, b := choose a, b : ("a and b are rational" and a < b < a + 1 and
                                (p*(a) > 0 and p*(b) < 0)
                             or (p*(a) < 0 and p*(b) > 0));
     while k < 2n do
          k := k + 1;
          d := choose d : ("d is rational" and (2a + b)/3 < d < (a + 2b)/3
                             and p*(d) ≠ 0);
          if (f(d) > 0 and f(a) > 0) or (f(d) < 0 and f(a) < 0)
               then a, b := d, b     { new bracket on right part of old }
               else a, b := a, d     { new bracket on left part of old }
          fi
     od;
     x := a                          { x := b would also work here }
end.
```

Figure 3

6. CONTINUITY OF COUNTABLY-MANY-VALUED *WhileCC** FUNCTIONS

In this section we define continuity for countably-many-valued functions, and then prove that countably-many-valued functions computed by ***WhileCC**** programs are continuous.

6.1 Topology and continuity with countably many values and '↑'

The results in this subsection are mostly of a technical nature, and their proofs are relegated to Appendix C. (Actually, all these results hold for arbitrary many-valued functions $f : X \to \mathcal{P}(Y)$, not necessarily countably-many-valued.) Recall Notation 4.2.1.

Definition 6.1.1 *(Totality).* The function $f : X \rightrightarrows Y$ is said to be *total* if for all $x \in X$, $f(x)$ is a *non-empty* subset of Y, *i.e.*, if $f : X \rightrightarrows^+ Y$.

Our semantic functions (in Section 7) will typically be of the form

$$\Phi : A^u \rightrightarrows^+ A^{v\uparrow}. \tag{10}$$

Remark 6.1.2. We think of the "deterministic version" of (10) as being a total function Φ, where for each $x \in X$, $\Phi(x)$ is a *singleton*, containing either an element of A^v (to indicate convergence) or '↑' (to indicate divergence). (*Cf.* Remark 4.2.2.)

We must now consider what it means for such a function (10) to be *continuous*.

Definition 6.1.3 *(Continuity).* Let $f: X \rightrightarrows Y$, for topological spaces X, Y.
(a) For any $V \subseteq Y$,

$$f^{-1}[V] \;=_{df}\; \{\, x \in X \mid f(x) \cap V \neq \emptyset \,\},$$

i.e., $x \in f^{-1}[V]$ iff at least one of the elements of $f(x)$ lies in V.
(b) f is *continuous* (w.r.t. X and Y) iff for all open $V \subseteq Y$, $f^{-1}[V]$ is open in X.

Remarks 6.1.4. (a) For metric spaces X and Y, Definition 6.1.3(b) becomes: $f: X \rightrightarrows Y$ is continuous iff

$$\forall a \in X\, \forall b \in f(a)\, \forall \epsilon > 0\, \exists \delta > 0\, \forall x \in \mathbf{B}(a,\delta)\, \big(f(x) \cap \mathbf{B}(b,\epsilon) \neq \emptyset\big).$$

(b) Definition 6.1.3(b) reduces to the standard definition of continuity for total single-valued functions from X to Y.

(c) It also reduces to the definition of continuity for partial single-valued functions (Definition 3.5.1), as we will see below (Remark 6.1.9). We must first see how to extend the topology on Y to that on $Y^{\uparrow}$ (Definition 6.1.6 below).

Definition 6.1.5. For two functions $f: X \rightrightarrows Y$, $g: X \rightrightarrows Y$, we define

$$f \sqsubseteq g \quad \Longleftrightarrow_{df} \quad \text{for all } x \in X,\ f(x) \subseteq g(x).$$

Definition 6.1.6. We extend the topology on Y to $Y^{\uparrow}$ $(= Y \cup \{\uparrow\})$ by specifying that the only open set containing $\{\uparrow\}$ is $Y^{\uparrow}$. (So $Y^{\uparrow}$ is a "one-point compactification" of Y.)

Now, given a function $f: X \rightrightarrows Y^{\uparrow}$, we define functions

$$f^{\uparrow}: X \rightrightarrows Y^{\uparrow} \qquad \text{and} \qquad f^{-}: X \rightrightarrows Y$$

by

$$f^{\uparrow}(x) \;=\; f(x) \cup \{\uparrow\} \qquad \text{and} \qquad f^{-}(x) \;=\; f(x) \backslash \{\uparrow\}.$$

In other words, $f^{\uparrow}$ *adds* '$\uparrow$' to the set $f(x)$ for each $x \in X$ and f^{-} *removes* '$\uparrow$' from every such set. This changes the semantics of f (see Remark 4.2.2), but not its *continuity properties*, as will be seen from the following technical lemma, which will be used in the proof of continuity of computable functions below (§6.2).

LEMMA 6.1.7. *Let* $f: X \rightrightarrows Y$ *and* $g: X \rightrightarrows^{+} Y^{\uparrow}$ *be any two functions such that*

$$f \sqsubseteq g \sqsubseteq f^{\uparrow},$$

i.e., for all $x \in X$, $g(x) \neq \emptyset$, *and either* $g(x) = f(x)$ *or* $g(x) = f(x) \cup \{\uparrow\}$. *Then*

$$f \text{ is continuous} \iff g \text{ is continuous.}$$

COROLLARY 6.1.8. *Suppose* $f: X \rightrightarrows^{+} Y^{\uparrow}$ *(i.e.,* f *is total). Then*

$$f \text{ is continuous} \iff f^{-} \text{ is continuous} \iff f^{\uparrow} \text{ is continuous.}$$

Remark 6.1.9 *(Justification of Remark 6.1.4(c)).* Let $f : X \rightarrow Y$ be a single-valued partial function. Define

$$\check{f}: X \rightrightarrows Y \qquad \text{and} \qquad \hat{f}: X \rightrightarrows^{+} Y^{\uparrow}$$

by

$$\check{f}(x) = \begin{cases} \{ f(x) \} & \text{if } x \in \textbf{\textit{dom}}(f) \\ \emptyset & \text{otherwise} \end{cases} \qquad \text{and} \qquad \hat{f}(x) = \begin{cases} \{ f(x) \} & \text{if } x \in \textbf{\textit{dom}}(f) \\ \{ \uparrow \} & \text{otherwise.} \end{cases}$$

(We can view either $\check{f}$ or $\hat{f}$ as "representing" f in the present context, *cf.* Remark 6.1.2.) Then

$$\begin{array}{rll} & f & \text{is continuous (according to Def. 3.5.1)} \\ \Longleftrightarrow & \check{f} & \text{is continuous (according to Def. 6.1.3)} \\ \Longleftrightarrow & \hat{f} & \text{is continuous (according to Def. 6.1.3).} \end{array}$$

The equivalence of the continuity of f and $\check{f}$ follows immediately from the definitions. The equivalence of the continuity of $\check{f}$ and $\hat{f}$ follows from Lemma 6.1.7.

LEMMA 6.1.10. *Given* $f : X \rightrightarrows Y^{\uparrow}$, *extend it to* $\tilde{f} : X^{\uparrow} \rightrightarrows Y^{\uparrow}$ *by stipulating that* $\tilde{f}(\uparrow) = \uparrow$. *If* f *is continuous and total, then* $\tilde{f}$ *is continuous.*

Definition 6.1.11 *(Composition).* (*a*) Suppose $f: X \rightrightarrows Y$ and $g: Y \rightrightarrows Z$. We define $g \circ f: X \rightrightarrows Z$ by

$$(g \circ f)(x) = \bigcup \{ g(y) \mid y \in f(x) \}.$$

(*b*) Suppose $f: X \rightrightarrows Y^{\uparrow}$ and $g: Y \rightrightarrows Z^{\uparrow}$. We define $g \circ f: X \rightrightarrows^{+} Z^{\uparrow}$ by

$$(g \circ f)(x) = \bigcup \{ g(y) \mid y \in f(x) \cap Y \} \cup \{ \uparrow \mid \uparrow \in f(x) \}.$$

PROPOSITION 6.1.12 (CONTINUITY OF COMPOSITION).
(a) If $f: X \rightrightarrows Y$ *and* $g: Y \rightrightarrows Z$ *are continuous, then so is* $g \circ f: X \rightrightarrows Z$.
(b) If $f: X \rightrightarrows^{+} Y^{\uparrow}$ *and* $g: Y \rightrightarrows^{+} Z^{\uparrow}$ *are continuous, then so is* $g \circ f: X \rightrightarrows^{+} Z^{\uparrow}$.

Definition 6.1.13 *(Union of functions).* Let $f_i : X \rightrightarrows Y^{\uparrow}$ be a family of functions for $i \in I$. Then we define

$$\bigsqcup_{i \in I} f_i : X \rightrightarrows Y^{\uparrow}$$

by

$$(\bigsqcup_{i \in I} f_i)(x) = \bigcup_{i \in I} f_i(x).$$

LEMMA 6.1.14. *If* $f_i: X \rightrightarrows Y^{\uparrow}$ *is continuous for all* $i \in I$, *then so is* $\bigsqcup_{i \in I} f_i$.

6.2 Continuity of *WhileCC* computable functions

Let A be an N-standard topological Σ-algebra.

To prove that ***WhileCC**** procedures on A are continuous, we first prove that such procedures are (almost) equivalent to ***While*** procedures (without 'choose') in an extended signature, which includes a symbol $\mathsf{f}: \mathsf{nat} \rightarrow \mathsf{nat}$ for an "oracle function". Then we apply Lemma 6.1.7.

LEMMA 6.2.1 (ORACLE EQUIVALENCE LEMMA). *Given a* ***WhileCC***(Σ) *statement* S*, and procedure*

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

we can effectively construct a ***While***(Σ_{f}) *statement* S_{f} *and procedure*

$$P_{\mathsf{f}} \;\equiv\; \mathsf{func\ in\ a\ out\ b\ aux\ c\ begin}\ S_{\mathsf{f}}\ \mathsf{end}$$

in a signature Σ_{f} *which extends* Σ *by a function symbol* $\mathsf{f} : \mathsf{nat} \to \mathsf{nat}$*, such that, putting*

$$P^A_{\sqcup} \;=_{df}\; \bigsqcup_{f \in \mathcal{F}} P^A_f,$$

where $\mathcal{F} = \mathbb{N}^{\mathbb{N}}$ *is the set of all functions* $f : \mathbb{N} \to \mathbb{N}$ *and* P^A_f *is the interpretation of* P_{f} *in* A *formed by interpreting* f *as* f*, we have*

$$P^A \;\sqsubseteq\; P^A_{\sqcup} \;\sqsubseteq\; (P^A)^{\uparrow}. \tag{11}$$

(Recall Definition 6.1.5, and the definition of $P^A\colon A^u \rightrightarrows^+ A^{v\uparrow}$ in §4.2(g).)

PROOF. Intuitively, f represents a possible implementation of the 'choose' operator: $\mathsf{f}(n)$ is a possible value for the nth call of this operator in any particular implementation of P. We will then take the union of the interpretations over all such possible implementations.

In more detail: S_{f} is constructed from S as follows. Let c be a new "counter", *i.e.*, an auxiliary nat variable which is not in S. First, by "splitting up" assignments in S, and introducing more auxiliary nat variables, we re-write S in such a way that every occurrence of the 'choose' construct is in the context of an assignment of the form

$$\mathsf{z}' := \mathsf{choose\ z} : b \tag{12}$$

where the boolean term b does not contain the 'choose' construct. Now replace each assignment of the form (12) by the pair of assignments

$$\begin{array}{l}\mathsf{c} := \mathsf{c} + 1;\\ \mathsf{if}\ b\langle \mathsf{z}/\mathsf{f}(\mathsf{c})\rangle\ \mathsf{then}\ \mathsf{z}' := \mathsf{f}(\mathsf{c})\ \mathsf{else\ div}\end{array}$$

and initialise the value of c (at the beginning of the statement) to 0. The result is a ***While***$^*(\Sigma_{\mathsf{f}})$ procedure P_{f} with a body S_{f} which, for a given interpretation f of f, "interprets" successive executions of 'choose' by successive values of f, when this is possible (*i.e.*, $b\langle \mathsf{z}/f(\mathsf{c})\rangle$ has tt as one of its values), and otherwise, causes the execution to diverge.

For those f which (for a given input) always give "good" values for all the successive executions of 'choose' assignments (12) in S, P^A_f will give a possible implementation of P. For all other f, P^A_f will diverge. Since (for a given input) each P^A_f *either* simulates one possible implementation of successive executions of 'choose' in S *or* diverges, their "union" $P^A_{\sqcup}$ gives the result of *all* possible implementations of 'choose', plus divergence; hence the conclusion (11). □

THEOREM 6.2.2. *Let*

$$P \;\equiv\; \mathsf{func\ in\ a\ out\ b\ aux\ c\ begin}\ S\ \mathsf{end} \tag{13}$$

be a ***WhileCC*** *procedure, where* $\mathsf{a} : u$ *and* $\mathsf{b} : v$*. Then the interpretation*

$$P^A : A^u \rightrightarrows^+ A^{v\uparrow}$$

is continuous.

PROOF. In the notation of the Oracle Equivalence Lemma (6.2.1): P_f^A is continuous for all $f \in \mathcal{F}$, by the continuity theorem for ***While*** [Tucker and Zucker 2000, §6.5]. Hence $P_{\sqcup}^A$ is continuous, by Lemma 6.1.14. Hence, by (13) and Lemma 6.1.7, so is P^A. □

Remark 6.2.3. In the special case that P^A is deterministic, *i.e.*, single-valued:

$$P^A : A^u \rightharpoonup A^v,$$

it follows by Remark 6.1.9 that P^A is continuous according to our definition (3.5.1) of continuity for single-valued partial functions.

COROLLARY 6.2.4. *A* ***WhileCC**** *computable function on* A *is continuous.*

PROOF. Such a function is ***WhileCC*** computable on A^*, hence (by Theorem 6.2.2) continuous on A^*, and hence on A. □

6.3 Continuity of ***WhileCC**** approximable functions

Recall Definition 4.5.1.

THEOREM 6.3.1. *Let* A *be a metric* Σ*-algebra, and* $f : A^u \rightharpoonup A^v$*. If* f *is* ***WhileCC**** *approximable on* A *and* ***dom***(f) *is open in* A^u *then* f *is continuous.*

PROOF. Suppose f is approximable on A by the ***WhileCC**** procedure $P : \mathsf{nat} \times u \to v$. We will show that f is continuous, using Remark 3.5.2. Given $a \in$ ***dom***(f) and $\epsilon > 0$, choose N such that

$$2^{-N} < \epsilon/3. \tag{14}$$

Then by Definition 4.5.1,

$$\emptyset \neq P_N^A(a) \subseteq \mathbf{B}(f(a),\, 2^{-N}). \tag{15}$$

Choose $b \in P_N^A(a)$. By (15),

$$\mathsf{d}(f(a), b) < 2^{-N}. \tag{16}$$

By Corollary 6.2.4, P_N^A is continuous on A, and so by Remark 6.1.4(a), there exists $\delta > 0$ such that

$$\forall x \in \mathbf{B}(a, \delta),\ P_N^A(x) \cap \mathbf{B}(b, \epsilon/3) \neq \emptyset. \tag{17}$$

Since ***dom***(f) is open, we may assume that δ is small enough so that

$$\mathbf{B}(a, \delta) \subseteq \textbf{\textit{dom}}(f).$$

Take any $x \in \mathbf{B}(a, \delta)$. By Definition 4.5.1 again,

$$P_N^A(x) \subseteq \mathbf{B}(f(x),\, 2^{-N}) \tag{18}$$

By (17), choose $y \in P_N^A(x) \cap \mathbf{B}(b, \epsilon/3)$. So

$$\mathsf{d}(y, b) < \epsilon/3 \tag{19}$$

and by (18)

$$\mathsf{d}(f(x), y) \;<\; 2^{-N}. \tag{20}$$

Hence

$$\begin{array}{rcl} \mathsf{d}(f(x), f(a)) & \leq & \mathsf{d}(f(x), y) + \mathsf{d}(y, b) + \mathsf{d}(b, f(a)) \\ & < & \epsilon \end{array}$$

by (20), (19), (16) and (14). The theorem follows by Remark 3.5.2. □

7. CONCRETE COMPUTABILITY; SOUNDNESS OF $\boldsymbol{WhileCC^*}$ COMPUTATION ON COUNTABLE ALGEBRAS

To compute on a metric algebra A using a concrete model of computation, we choose a countable subspace X of A and an enumeration $\alpha : \mathbb{N} \to X$.

In this section we step back from topological algebras and consider computability on *arbitrary countable* algebras A. We show (Theorem A_0) that if A is enumerated by α and its basic functions are α-computable, then functions that are $\boldsymbol{WhileCC^*}$ computable on A are also α-computable. This is a key lemma in the soundness theorem for $\boldsymbol{WhileCC^*}$ approximation in the next section.

7.1 Enumerations and tracking functions for partial functions

Let $X = \langle X_s \mid s \in \boldsymbol{Sort}(\Sigma)\rangle$ be a family of non-empty sets, indexed by $\boldsymbol{Sort}(\Sigma)$.

Definition 7.1.1. An *enumeration* of X is a family

$$\alpha \;=\; \langle \alpha_s \colon \Omega_s \twoheadrightarrow X_s \mid s \in \boldsymbol{Sort}(\Sigma)\rangle$$

of surjective maps $\alpha_s \colon \Omega_s \twoheadrightarrow X_s$, for some family

$$\Omega \;=\; \langle \Omega_s \mid s \in \boldsymbol{Sort}(\Sigma)\rangle$$

of sets $\Omega_s \subseteq \mathbb{N}$. The family X is said to be *enumerated by* α. We say that $\alpha \colon \Omega \twoheadrightarrow X$ is an *enumeration* of X, and call the pair (X, α) an *enumerated family of sets*. (The notation '$\twoheadrightarrow$' denotes surjections, or onto mappings.)

We also write $\Omega_{\alpha,s}$ for Ω_s to make explicit the fact that $\Omega_s = \boldsymbol{dom}(\alpha_s)$.

Definition 7.1.2 *(Tracking and strict tracking functions).* We use the notation $X^u = X_{s_1} \times \cdots \times X_{s_m}$ and $\Omega_\alpha^u = \Omega_{\alpha,s_1} \times \cdots \times \Omega_{\alpha,s_m}$, where $u = s_1 \times \cdots \times s_m$. Let $f \colon X^u \dot{\to} X_s$ and $\varphi : \Omega_\alpha^u \dot{\to} \Omega_{\alpha,s}$,

(*a*) φ is a *tracking function with respect to* α, or α*-tracking function*, for f, if the following diagram commutes:

$$\begin{array}{ccc} X^u & \xrightarrow{\;f\;} & X_s \\ {\scriptstyle \alpha^u}\uparrow & & \uparrow{\scriptstyle \alpha_s} \\ \mathbb{N}^m & \xrightarrow[\;\varphi\;]{} & \mathbb{N} \end{array}$$

in the sense that for all $k \in \Omega_\alpha^u$

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

(*b*) φ is a *strict α-tracking function* for f if in addition, for all $k \in \Omega_\alpha^u$

$$f(\alpha^u(k))\!\uparrow \quad \Longrightarrow \quad \varphi(k)\!\uparrow.$$

Here we use the notation $\alpha^u(k) = (\alpha_{s_1}(k_1), \ldots, \alpha_{s_m}(k_m))$, where $k = (k_1, \ldots, k_m)$. (We will sometimes drop the type super- and subscripts.)

Definition 7.1.3 *(α-computability).* (*a*) Suppose A is a $\boldsymbol{Sort}(\Sigma)$-family, and (X, α) an enumerated subfamily of A, *i.e.*, $X_s \subseteq A_s$ for all Σ-sorts s. Suppose we have

$$f\colon A^u \rightharpoonup A_s \qquad \text{and} \qquad \varphi : \mathbb{N}^m \rightharpoonup \mathbb{N}$$

such that

$$f \restriction X^u\colon X^u \;\dot{\to}\; X_s \qquad \text{and} \qquad \varphi \restriction \Omega_\alpha^u\colon \Omega_\alpha^u \;\dot{\to}\; \Omega_{\alpha,s},$$

and $\varphi \restriction \Omega_\alpha^u$ is a (*strict*) *α-tracking function* for $f \restriction X$. We then say that φ is a (*strict*) *α-tracking function for f.*

(*b*) Suppose now further that φ is a *computable* (*i.e.*, recursive) partial function. Then f is said to be (*strictly*) *α-computable.*

Remarks 7.1.4. (*a*) In the situation of Definition 7.1.3, we are not concerned with the behaviour of f off X^u, or the behaviour of φ off Ω_α^u.

(*b*) For total f, the concepts of tracking function and strict tracking function coincide; as do the concepts of α-computability and strict α-computability.

(*c*) For convenience, we will always assume:

$$\Omega_{\alpha,\mathsf{bool}} = \{0, 1\}, \qquad \alpha_{\mathsf{bool}}(0) = \mathsf{ff}, \qquad \alpha_{\mathsf{bool}}(1) = \mathsf{tt}$$

and also (when Σ is N-standard):

$$\Omega_{\alpha,\mathsf{nat}} = \mathbb{N} \text{ and } \alpha_{\mathsf{nat}} \text{ is the identity on } \mathbb{N}.$$

Assume now that A is a *Σ-algebra* and (X, α) is a $\boldsymbol{Sort}(\Sigma)$-family of subsets of A, enumerated by α.

Definition 7.1.5 *(Enumerated Σ-subalgebra).* (X, α) is said to be an *enumerated Σ-subalgebra of* A if X is a Σ-subalgebra of A.

Definition 7.1.6 *(Σ-effective subalgebra).* Suppose A is a Σ-algebra and (X, α) is an enumerated Σ-subalgebra. Then α is said to be

(*a*) *Σ-effective* if all the basic Σ-functions on A are α-computable; and

(*b*) *strictly Σ-effective* if all the basic Σ-functions on A are strictly α-computable.

7.2 Soundness Theorem for surjective enumerations

For the rest of this section we will be considering the special case of §7.1 in which the enumerated subalgebra X is A itself, *i.e.*, we assume the enumeration is *onto* A. To emphasise this special situation, we will denote the enumeration by

$$\beta\colon \Omega_\beta \;\twoheadrightarrow\; A,$$

so that (A, β) is our *enumerated Σ-algebra.* Then given a function

$$f\colon A^u \;\dot{\to}\; A_s,$$

we have two notions of computability for f:

(*i*) *abstract*, *i.e.*, ***WhileCC**** computability, as described in Section 4; and

(*ii*) *concrete*, *i.e.*, β-computability, as in Definition 7.1.3, in the special case that $X = A$.

We will prove a *soundness theorem* (Theorem A_0), for these notions of abstract and concrete computability, *i.e.*, (*i*)⇒(*ii*), assuming *strict effectiveness* of β.

A more general soundness theorem (Theorem A), with more general notions of abstract computability (***WhileCC**** *approximability*) and concrete computability (computability w.r.t. the *computable closure* of an enumeration), will be proved in Section 8.

THEOREM A_0 (SOUNDNESS FOR COUNTABLE ALGEBRAS). *Let* (A, β) *be an enumerated N-standard* Σ*-algebra such that* β *is strictly* Σ*-effective. If* $f : A^u \to A_s$ *is* ***WhileCC**** *computable on* A*, then* f *is strictly* β*-computable on* A*.*

7.3 Proof of Soundness Theorem A_0

Assume, then, that (A, β) is an enumerated N-standard Σ-algebra and β is strictly Σ-effective. We must show that each of the semantic functions listed in §4.2(*a*)–(*g*) has a computable strict tracking function. More precisely, we work, not with the semantic functions themselves, but "localised" functions representing them [Tucker and Zucker 2000, §4]. This amounts to proving a series of results of the form:

LEMMA SCHEME 7.3.1. *For each* ***WhileCC*** *semantic representing function*

$$\Phi : \ A^u \ \rightrightarrows^+ \ A^{v\uparrow}$$

representing one of the semantic functions listed in §4.2(a)–(g), there is a computable tracking function w.r.t. β*, i.e., a function*

$$\varphi : \ \Omega^u_\beta \ \dot{\to} \ \Omega^v_\beta$$

which commutes the diagram

$$\begin{array}{ccc} A^u & \overset{\Phi}{\Longrightarrow} & {}^+A^{v\uparrow} \\ {\scriptstyle\beta^u}\uparrow & & \uparrow{\scriptstyle\beta^v} \\ \Omega^u_\beta & \underset{\varphi}{\dot{\longrightarrow}} & \Omega^v_\beta \end{array}$$

in the sense that for all $k, l \in \Omega^u_\beta$*:*

$$\begin{array}{lcl} \varphi(k)\downarrow l & \Longrightarrow & \beta^v(l) \ \in \ \Phi(\beta^u(k)), \\ \varphi(k)\uparrow & \Longrightarrow & \uparrow \ \in \ \Phi(\beta^u(k)). \end{array}$$

[This Lemma Scheme is proved in Appendix D.]

Remarks 7.3.2. (*a*) Here φ is a combination "strict tracking function" and "selection function". We can think of φ as giving one possible implementation of Φ. (Compare the representative functions for various semantic functions in [Tucker and Zucker 2000, §4].)

(*b*) We are not concerned with the behaviour of φ on $\mathbb{N}^m \backslash \Omega^u_\beta$. (*Cf.* Remark 7.1.4(*a*).)

Theorem A_0 then follows easily from this lemma scheme. (See Appendix D).

8. SOUNDNESS OF $\boldsymbol{WhileCC}^*$ APPROXIMATION

We return to the general situation introduced in §7.2, of a partial metric Σ-algebra A with an enumerated subalgebra (X, α), and prove a more general soundness theorem (Theorem A) for $\boldsymbol{WhileCC}^*$ approximation. From the enumeration $\alpha : \mathbb{N} \to X$ we will build the space $\mathsf{C}_\alpha(X)$ of α-computable elements of A, and enumerate it with $\overline{\alpha} : \mathbb{N} \to\!\!\!\!\cdot\;\; \mathsf{C}_\alpha(X)$.

8.1 Enumerated subspace of metric algebra; Computational closure

Let A be an N-standard metric Σ-algebra, and (X, α) an enumerated $\boldsymbol{Sort}(\Sigma)$-family $\langle (X_s, \alpha) \mid s \in \boldsymbol{Sort}(\Sigma) \rangle$ of subsets $X_s \subseteq A_s$ $(s \in \boldsymbol{Sort}(\Sigma))$. Each X_s can be viewed as a *metric subspace* of the metric space A_s. We call (X, α) a $\boldsymbol{Sort}(\Sigma)$-*enumerated (metric) subspace* of A. From (X, α) we define a family

$$\mathsf{C}_\alpha(X) \;=\; \langle \mathsf{C}_\alpha(X)_s \mid s \in \boldsymbol{Sort}(\Sigma) \rangle$$

of sets $\mathsf{C}_\alpha(X)_s$ of *α-computable elements of A_s*, *i.e.*, limits in A_s of effectively convergent Cauchy sequences (to be defined below) of elements of X_s, so that

$$X_s \;\subseteq\; \mathsf{C}_\alpha(X)_s \;\subseteq\; A_s,$$

with corresponding enumerations

$$\overline{\alpha}_s : \; \Omega_{\overline{\alpha},s} \;\twoheadrightarrow\; \mathsf{C}_\alpha(X)_s.$$

Writing $\overline{\alpha} = \langle \overline{\alpha}_s \mid s \in \boldsymbol{Sort}(\Sigma) \rangle$, we call the enumerated subspace $(\mathsf{C}_\alpha(X), \overline{\alpha})$ the *computable closure* of (X, α) in A.

We will generally be interested in $\overline{\alpha}$-computable (rather than α-computable) functions on A (*cf.* Definition 7.1.3), as our model of *concrete computability* on A.

The sets $\Omega_{\overline{\alpha},s} \subseteq \mathbb{N}$ consist of *codes* for $\mathsf{C}_\alpha(X)_s$ (w.r.t. α), *i.e.*, pairs of numbers $c = \langle e, m \rangle$ where

(*i*) e is an index for a total recursive function defining a sequence $\alpha \circ \{e\}$ in X_s, *i.e.*, the sequence

$$\alpha_s(\{e\}(0)),\; \alpha_s(\{e\}(1)),\; \alpha_s(\{e\}(2)),\; \ldots\;, \tag{21}$$

of elements of X_s,

(*ii*) m is an index for a modulus of convergence for this sequence:

$$\forall k, l \geq \{m\}(n) : \; \mathsf{d}_s(\alpha(\{e\}(k)), \alpha(\{e\}(l))) < 2^{-n}. \tag{22}$$

For any such code $c = \langle e, m \rangle \in \Omega_{\overline{\alpha},s}$, $\overline{\alpha}_s(c)$ is defined as the limit in A_s of the Cauchy sequence (21), and $\mathsf{C}_\alpha(X)_s$ is the range of $\overline{\alpha}_s$:

$$\begin{array}{ccccc} X_s & \subseteq & \mathsf{C}_\alpha(X)_s & \subseteq A \\ {\scriptstyle \alpha_s}\uparrow & & {\scriptstyle \overline{\alpha}_s}\uparrow & \\ \Omega_{\alpha,s} & & \Omega_{\overline{\alpha},s} & \end{array}$$

Remarks 8.1.1. (*a*) (*Fast Cauchy sequences.*) We may assume, when convenient, that the modulus of convergence for a given code is the *identity*, *i.e.*, replace (22) by the simpler

$$\forall k, l \geq n : \ \mathsf{d}_s(\alpha(\{e\}(k)), \alpha(\{e\}(l))) < 2^{-n}$$

or, equivalently,

$$\forall k > n : \ \mathsf{d}_s(\alpha(\{e\}(k)), \alpha(\{e\}(n))) < 2^{-n}, \tag{23}$$

because any code $c = \langle e, m\rangle$ satisfying (22) can be effectively replaced by a code for the same element of $\mathsf{C}_\alpha(X)_s$ satisfying (23), namely $c' = \langle e', m_1\rangle$, where m_1 is a standard code for the identity function on $\mathbb{N}$, and $e' = \mathrm{comp}(e, m)$, where $\mathrm{comp}(x, y)$ is a primitive recursive function for "composition" of (indices of) computable functions, *i.e.*, $\{\,\mathrm{comp}(e, m)\,\}(x) \simeq \{\,e\,\}(\{\,m\,\}(x))$. In the case of a code $c = \langle e, m_1\rangle$ satisfying (23), the sequence (21) is called a *fast (α-effective) Cauchy sequence*. We may then, for simplicity, call e itself the "code", and the argument of $\overline{\alpha}_s$. So we can shift between "c-codes" and "e-codes" as convenient.

(*b*) In the case $s = \mathsf{nat}$, we can simply take $\Omega_{\overline{\alpha},\mathsf{nat}} = \Omega_{\alpha,\mathsf{nat}} = \mathbb{N}$, and $\overline{\alpha}_{\mathsf{nat}}$ and α_{nat} as the identity mappings on $\mathbb{N}$. Similarly, in the case $s = \mathsf{bool}$, we can take $\Omega_{\overline{\alpha},\mathsf{bool}} = \Omega_{\alpha,\mathsf{bool}} = \{\,0, 1\,\}$, with $\overline{\alpha}(0) = \alpha(0) = \mathsf{ff}$ and $\overline{\alpha}(1) = \alpha(1) = \mathsf{tt}$. (*Cf.* Remark 7.1.4(*c*).)

(*c*) (*Closure of α-computability operation*) The subspace $(\mathsf{C}_\alpha(X), \overline{\alpha})$ is "computationally closed in A", in the sense that the limit of a (fast) $\overline{\alpha}$-effective Cauchy sequence of elements of $\mathsf{C}_\alpha(X)$ is again in $\mathsf{C}_\alpha(X)$, *i.e.*, $\mathsf{C}_{\overline{\alpha}}(\mathsf{C}_\alpha(X)) = \mathsf{C}_\alpha(X)$. (*Easy exercise.*)

(*d*) (*Decidability of $\Omega_{\alpha,s}$*) We usually assume that $\Omega_{\alpha,s}$ is decidable, in fact, that $\Omega_{\alpha,s} = \mathbb{N}$ for all s, which is typical in practice, unlike the case for $\Omega_{\overline{\alpha}}$. (See Example 8.1.2.)

(*e*) (*Extension of enumeration to A^**) Given an enumeration α of a Σ-subspace X of A, we can extend this canonically to an enumeration α^* of a Σ^*-subspace X^* of A^*. (*Easy exercise.*) This in turn generates an enumeration $\overline{\alpha^*}$ of a Σ^*-subspace $\mathsf{C}_\alpha(X)^*$ of α^*-computable elements of A^*. It is easy to see that

(*i*) if $\mathsf{C}_\alpha(X)$ is an Σ-subalgebra of A, then $\mathsf{C}_\alpha(X)^*$ is a Σ^*-subalgebra of A^*;
(*ii*) if $\overline{\alpha}$ is (strictly) Σ-effective, then $\overline{\alpha^*}$ is (strictly) Σ^*-effective.

We will usually use this extension (of (X, α) and $(\mathsf{C}_\alpha(X), \overline{\alpha})$) to A^* implicitly, *i.e.*, writing 'α' instead of 'α^*' etc.

Example 8.1.2 *(Constructible reals).* The best known nontrivial example of an enumerated subspace (X, α), and its extension to a subspace of α-computable elements, is the following. Let A be the metric algebra $\mathcal{R}_p$ of reals (Example 3.6.1), with signature Σ. Let X_{real} be the set of rationals $\mathbb{Q} \subset \mathbb{R}$, let $\Omega_{\alpha,\mathsf{real}} = \mathbb{N}$ and let $\alpha_{\mathsf{real}} : \mathbb{N} \to \mathbb{Q}$ be a canonical enumeration of $\mathbb{Q}$. Then $\mathsf{C}_\alpha(\mathbb{Q}) =_{df} \mathsf{C}_\alpha(X)_{\mathsf{real}} \subset \mathbb{R}$ is the subspace of *recursive* or *constructible reals*. Note that it is a *subfield* of $\mathbb{R}$, and hence $\mathsf{C}_\alpha(X)$ is a *subalgebra* of $\mathcal{R}$. Further, it is easily verified that $\overline{\alpha}$ is strictly $\Sigma(\mathcal{R})$-effective. (*Cf.* Definition 7.1.6.) Note that $\Omega_{\alpha,\mathsf{real}} = \mathbb{N}$, whereas $\Omega_{\overline{\alpha},\mathsf{real}}$ is non-recursive. (See Remark 8.1.1(*d*).)

8.2 Soundness theorem for effective numberings

We now prove the first main theorem mentioned in the Introduction.

THEOREM A (SOUNDNESS). *Let A be an N-standard metric Σ-algebra, and (X, α) an enumerated $\boldsymbol{Sort}(\Sigma)$-subspace. Suppose the enumerated $\boldsymbol{Sort}(\Sigma)$-space $(\mathsf{C}_\alpha(X), \overline{\alpha})$ of α-computable elements of A is a Σ-subalgebra of A, and $\overline{\alpha}$ is strictly Σ-effective. If $f : A^u \rightarrow A_s$ is $\boldsymbol{WhileCC}^*$-approximable on A, then f is $\overline{\alpha}$-computable on A.*

PROOF. The proof uses the Soundness Theorem A_0 (Section 7), or rather the Lemma Scheme 7.3.1 (specifically, part (g) of the proof) applied to the enumerated subalgebra $(\mathsf{C}_\alpha(X), \overline{\alpha})$ in place of (A, β).

So suppose $f : A^u \rightarrow A_s$ is effectively uniformly $\boldsymbol{WhileCC}^*$ approximable on A. Then there is a $\boldsymbol{WhileCC}^*(\Sigma)$ procedure

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

such that for all $n \in \mathbb{N}$ and all $x \in \boldsymbol{dom}(f)$:

$$\uparrow \notin P_n^A(x) \subseteq \mathbf{B}(f(x), 2^{-n}) \tag{24}$$

(see Definition 4.5.1). By Lemma Scheme 7.3.1 (specifically, part (g) of the proof, applied to $(\mathsf{C}_\alpha(X), \overline{\alpha})$ in place of (A, β)) there is a computable function

$$\psi : \mathbb{N} \times \Omega_{\overline{\alpha}}^u \rightarrow \Omega_{\overline{\alpha},s}$$

which tracks P^A strictly, in the sense that for all $n \in \mathbb{N}$, $e \in \Omega_{\overline{\alpha}}^u$ and $e' \in \Omega_{\overline{\alpha},s}$ (and writing $\psi_n = \psi(n, \cdot)$):

$$\begin{array}{lll} \psi_n(e) \downarrow e' & \Longrightarrow & \overline{\alpha}(e') \in P_n^A(\overline{\alpha}(e)), \\ \psi_n(e) \uparrow & \Longrightarrow & \uparrow \in P_n^A(\overline{\alpha}(e)). \end{array} \tag{25}$$

We will show how to define a partial recursive $\overline{\alpha}$-tracking function

$$\varphi : \Omega_{\overline{\alpha}}^u \rightarrow \Omega_{\overline{\alpha},s}$$

for f as follows. Given any $e \in \Omega_{\overline{\alpha}}^u$, suppose $\overline{\alpha}(e) \in \boldsymbol{dom}(f)$, *i.e.*,

$$f(\overline{\alpha}(e)) \downarrow \in A_s. \tag{26}$$

We must show how to define an $\overline{\alpha}$-tracking function φ for f, *i.e.*, such that

$$\varphi(e) \in \Omega_{\overline{\alpha},s} \qquad \text{and} \qquad \overline{\alpha}(\varphi(e)) = f(\overline{\alpha}(e)). \tag{27}$$

By (24), for all n

$$\uparrow \notin P_n^A(\overline{\alpha}(e)) \subseteq \mathbf{B}(f(\overline{\alpha}(e)), 2^{-n}). \tag{28}$$

Hence by (24), for all n

$$\psi_n(e) \downarrow \in \Omega_{\overline{\alpha},s} \tag{29}$$

and

$$\overline{\alpha}(\psi_n(e)) \in P_n^A(\overline{\alpha}(e)). \tag{30}$$

and so by (29) we may assume (by definition of $\Omega_{\overline{\alpha}}$) that for all n

$$\alpha \circ \{\psi_n(e)\} \text{ is a fast Cauchy sequence, with limit } \overline{\alpha}(\psi_n(e)). \tag{31}$$

Also by (29) and (28),

$$\mathsf{d}\big(\overline{\alpha}(\psi_n(e)),\ f(\overline{\alpha}(e))\big)\ <\ 2^{-n}. \tag{32}$$

Now let e' be a "canonical" index for the (partial) function

$$\{e'\}:\ n\ \mapsto \{\psi_n(e)\}(n) \tag{33}$$

obtained uniformly effectively in e. So $\{e'\}$ is the "diagonal" function formed from the sequence of functions with indices $\psi_n(e)$. Consider the sequence $\alpha_s \circ \{e'\}$, *i.e.*,

$$\alpha_s(\{e'\}(0)),\ \alpha_s(\{e'\}(1)),\ \alpha_s(\{e'\}(2)),\ \ldots\ . \tag{34}$$

CLAIM: (34) is a Cauchy sequence in A_s, with modulus of convergence $\lambda n(n+2)$.

PROOF OF CLAIM: For any n and $k > n$:

$$\begin{array}{ll} & \mathsf{d}\big(\alpha(\{e'\}(k)),\ \alpha(\{e'\}(n)\big) \\ = & \mathsf{d}\big(\alpha(\{\psi_k(e)\}(k),\ \alpha(\{\psi_n(e)\}(n)\big) \qquad \text{by def. (9) of } e' \\ \leq & \mathsf{d}\big(\alpha(\{\psi_k(e)\}(k)), \overline{\alpha}(\psi_k(e))\big)\ +\ \mathsf{d}\big(\overline{\alpha}(\psi_k(e)), \overline{\alpha}(\psi_n(e))\big)\ +\ \mathsf{d}\big(\overline{\alpha}(\psi_n(e)), \alpha(\{\psi_n(e)\}(n))\big) \\ = & \mathsf{d}_1\ +\ \mathsf{d}_2\ +\ \mathsf{d}_3 \qquad \text{(say)} \end{array}$$

where by (31)

$$\mathsf{d}_1\ \leq\ 2^{-k}, \qquad\qquad \mathsf{d}_3\ \leq\ 2^{-n},$$

and by (32)

$$\begin{array}{ll} \mathsf{d}_2 & \leq\ \mathsf{d}\big(\overline{\alpha}(\psi_k(e)),\ f(\overline{\alpha}(e))\big)\ +\ \mathsf{d}\big(f(\overline{\alpha}(e)),\ \overline{\alpha}(\psi_n(e))\big) \\ & <\ 2^{-k}\ +\ 2^{-n}. \end{array}$$

Therefore

$$\begin{array}{lll} \mathsf{d}\big(\alpha(\{e'\}(k)),\ \alpha(\{e'\}(n)\big) & \leq & \mathsf{d}_1\ +\ \mathsf{d}_2\ +\ \mathsf{d}_3 \\ & < & 2\cdot 2^{-k}\ +\ 2\cdot 2^{-n} \\ & < & 2^{-n+2}. \end{array}$$

This proves the claim. □

Further, by the method of Remark 8.1.1(a) (composing $\{e'\}$ with the modulus of convergence), we can replace the index e' by an e-code e'' for a fast Cauchy sequence:

$$\{e''\}(n)\ \simeq\ \{e'\}(n+2). \tag{35}$$

Then we define

$$\varphi(e)\ =\ e''. \tag{36}$$

We show that φ is an $\overline{\alpha}$-tracking function for f, *i.e.*, (assuming (26)) we show (27). Since $\alpha \circ \{e''\}$ is a fast Cauchy sequence, with the same limit in A (if it exists) as $\alpha \circ \{e'\}$ (by its definition (35)), to prove (24) it is enough to show (by (24)) that

$$\alpha(\{e'\}(n))\ \to\ f(\overline{\alpha}(e)) \qquad \text{as} \qquad n\ \to\ \infty. \tag{37}$$

This follows since

$$\begin{array}{lll} \mathsf{d}\big(\alpha(\{e'\}(n),\ f(\overline{\alpha}(e))\big) & = & \mathsf{d}\big(\alpha(\{\psi_n(e)\}(n)),\ f(\overline{\alpha}(e))\big) \qquad\qquad \text{by def. (9) of } e' \\ & \leq & \mathsf{d}\big(\alpha(\{\psi_n(e)\}(n)),\ \overline{\alpha}(\psi_n(e))\big)\ +\ \mathsf{d}\big(\overline{\alpha}(\psi_n(e)),\ f(\overline{\alpha}(e))\big) \\ & < & 2^{-n}\ +\ 2^{-n} \qquad\qquad \text{by (31) and (32)} \\ & = & 2^{-n+1} \end{array}$$

proving (24). □

A deterministic version of Theorem A (*i.e.*, without 'choose') was proved in [Stewart 1998].

9. ADEQUACY OF ***WhileCC**** APPROXIMATION

9.1 Adequacy Theorem

In this section we will prove Theorem B, a converse to the result of the previous section. Assume that A is an N-standard metric Σ-algebra, and (X, α) an enumerated Σ-subspace, with α-computable closure $(\mathsf{C}_\alpha(X), \overline{\alpha})$.

Note that we are not assuming in this section that $\mathsf{C}_\alpha(X)$ is a subalgebra of A, or even that $\overline{\alpha}$ is Σ-effective.

One of the assumptions in the theorem, "effective local uniform continuity w.r.t. an open exhaustion", must first be defined, as must "open exhaustion".

Definition 9.1.1 *(Open exhaustion).* Let U be a subset of a metric space X. An *open exhaustion of* U is a sequence of open subsets of X

$$V \;=\; (V_0, V_1, V_2, \dots) \qquad \text{such that} \qquad \bigcup_{p=0}^{\infty} V_p \;=\; U.$$

Remarks 9.1.2. (*a*) Clearly, if U has an open exhaustion, then U is open.

(*b*) It is helpful (though not necessary) to think of the sets of an exhaustion as increasing: $V_0 \subseteq V_1 \subseteq V_2 \subseteq \dots$.

(*c*) Any open set U has the *trivial exhaustion* $U, U, \dots$.

(*d*) A simple non-trivial example of an open exhaustion is the *standard open exhaustion* V of $\mathbb{R}$, where $V_p = (-p, p)$.

We also need an effective notion of open exhaustion:

Definition 9.1.3. An open exhaustion V of $U \subseteq A^u$ is ***WhileCC***-effective* in A if it satisfies the following two conditions:

(*a*) (***WhileCC***-effective Archimedean property of U w.r.t. V*) There is a ***WhileCC**** procedure $P_{\mathsf{loc}} : u \to \mathsf{nat}$ which, given $x \in U$, "locates" x in V, *i.e.*, produces some p such that $x \in V_p$; more precisely:

$$P^A_{\mathsf{loc}}(x) \;=\; \begin{cases} \{\, p \mid x \in V_p \,\} & \text{if } \mathrm{x} \in U \\ \{\uparrow\} & \text{otherwise.} \end{cases}$$

(*b*) (***WhileCC***-effective openness* of V) There is a ***WhileCC****-computable function $\gamma : A_u \times \mathbb{N} \to \mathbb{N}$ such that for all p and all $x \in V_p$,

$$\mathbf{B}(x,\, 2^{-\gamma(x,p)}) \;\subseteq\; V_p.$$

Remarks 9.1.4. (*a*) Typically, the procedure $P_{\mathsf{loc}}(x)$ is realised in the form "$\mathsf{choose}\ p : x \in V_p$" where "$x \in V_p$" can formalised as a boolean test in the language.

(*b*) The standard open exhaustion of $\mathbb{R}$ (Remark 9.1.2(*d*)) is ***WhileCC****-effective in $\mathcal{R}^N_p$.

Definition 9.1.5 *(Effective global and local uniform continuity).* Let X and Y be metric spaces, and let $f : X \dot{\to} Y$.

(a) We say f is *effectively (globally) uniformly continuous* iff $\boldsymbol{dom}(f)$ is open and there is a recursive function $\delta : \mathbb{N} \to \mathbb{N}$ such that for all n and all $x, y \in \boldsymbol{dom}(f)$:

$$\mathsf{d}_X(x,y) < 2^{-\delta(n)} \implies \mathsf{d}_Y(f(x), f(y)) < 2^{-n}$$

(b) We say f is *effectively locally uniformly continuous* w.r.t. an open exhaustion V of $\boldsymbol{dom}(f)$ iff there is a recursive $\delta : \mathbb{N}^2 \to \mathbb{N}$ such that for all p, n and all $x, y \in V_p$:

$$\mathsf{d}_X(x,y) < 2^{-\delta(p,n)} \implies \mathsf{d}_Y(f(x), f(y)) < 2^{-n}.$$

Example 9.1.6. This occurs typically when A is a countable union of neighbourhoods with compact closure; for example, in the algebra $\mathcal{R}_p$ of reals, $\mathbb{R}$ is the union of the neighbourhoods $(-k,\, k)$ for $k = 1, 2, \ldots$. Then a continuous function f on $\mathbb{R}$ will be uniformly continuous on each of these neighbourhoods.

Remarks 9.1.7. (a) Effective global and local uniform continuity implies continuity (as we would hope).

(b) Effective global uniform continuity of f corresponds to the special case of effective local uniform continuity with respect to the *trivial exhaustion* of $\boldsymbol{dom}(f)$.

We are now ready for the theorem.

THEOREM B (ADEQUACY). *Let A be an N-standard metric Σ-algebra, (X, α) an enumerated $\boldsymbol{Sort}(\Sigma)$-subspace, and $(\mathsf{C}_\alpha(X), \overline{\alpha})$ the $\boldsymbol{Sort}(\Sigma)$-subspace of α-computable elements of A. Suppose that for all Σ-sorts s:*

(i) X_s is dense in A_s, and

(ii) $\alpha_s : \mathbb{N} \to A_s$ is $\boldsymbol{WhileCC}^*$-computable on A.

Let $f : A^u \dot{\to} A_s$ be a function on A and V an open exhaustion of $\boldsymbol{dom}(f)$ such that

(iii) V is $\boldsymbol{WhileCC}^*$-effective, and

(iv) f is effectively locally uniformly continuous w.r.t. V.

If f is $\overline{\alpha}$-computable on A, then f is $\boldsymbol{WhileCC}^*$ approximable on A.

Remark 9.1.8. From the proof of the theorem, it will be apparent that only sorts s in the domain of f have to satisfy condition (i), and only sorts s in the domain or range of f have to satisfy condition (ii).

The proof uses the following notation.

Notation 9.1.9 *(Embedding X into its $\boldsymbol{\alpha}$-computational closure).* By elementary recursion theory, there is a primitive recursive function $\mathsf{const} : \mathbb{N} \to \mathbb{N}$ such that for each k, $\mathsf{const}(k)$ is the index of the function on $\mathbb{N}$ with constant value k, *i.e.*, for all n,

$$\{\mathsf{const}(k)\}(n) \;=\; k,$$

Thus for all k, $\mathsf{const}(k)$ can be taken as a code for a fast Cauchy sequence in X (see Remark 8.1.1(a)), making const an $\alpha, \overline{\alpha}$-tracking function for the inclusion map

$\iota\colon X \hookrightarrow \mathsf{C}_\alpha(X)$, in the sense that for each sort s the following diagram commutes:

$$\begin{array}{ccc} X_s & \xrightarrow{\iota_s} & \mathsf{C}_\alpha(X)_s \\ {\scriptstyle \alpha_s}\uparrow & & \uparrow{\scriptstyle \overline{\alpha}_s} \\ \mathbb{N} & \xrightarrow[\mathsf{const}]{} & \Omega_{\overline{\alpha},s} \end{array}$$

9.2 Proof of Theorem B

We give (in ***WhileCC**** pseudo-code) an algorithm for a function

$$g:\ \mathbb{N} \times A^u \ \rightrightarrows^+\ A_s{}^{\uparrow}$$

which approximates f, in the sense that for all n and all $x \in \boldsymbol{dom}(f)$,

$$g_n(x) \ \subseteq\ \mathbf{B}(f(x),\, 2^{-n}) \ \subseteq\ A_s. \tag{38}$$

With input n and $x \in A^u$: assume $x \in \boldsymbol{dom}(f)$ (otherwise we don't care about the output).

(1°) First, we want to find some p such that $x \in V_p$. This is ***WhileCC**** computable, by the ***WhileCC****-effectiveness of V (assumption (iii)). Note the use of the 'choose' construct in "finding" p (see Remark 9.1.4), even though p will not be an explicit argument of g. Note that (still by (iii), and in the notation of Definition 9.1.3)

$$\mathbf{B}(x,\ 2^{-\gamma(x,p)}) \ \subseteq\ V_p \ \subseteq\ \boldsymbol{dom}(f). \tag{39}$$

Now, using assumption (iv) and in the notation of Definition 9.1.5(b), compute

$$M \ :=\ \max(\gamma(x,p),\ \delta(p, n+1)) \tag{40}$$

which (since γ is ***WhileCC**** computable and δ is recursive) is ***WhileCC**** computable.

(2°) Next we want to find some k such that

$$\mathsf{d}(\alpha(k),\ x) \ <\ 2^{-M}. \tag{41}$$

By the density assumption (i) such a k exists. Again, we can find such a k using the 'choose' construct. Note again the use of the 'choose' construct in "finding" k, even though k will not be an explicit argument of g. Now by (40) and (39),

$$\mathbf{B}(x,\ 2^{-M}) \ \subseteq\ \mathbf{B}(x,\ 2^{-\gamma(x,p)}) \ \subseteq\ \boldsymbol{dom}(f)$$

and so by (41)

$$\overline{\alpha}(\mathsf{const}(k)) \ =\ \alpha(k) \ \in\ \boldsymbol{dom}(f). \tag{42}$$

By assumption, f has an $\overline{\alpha}$-tracking function φ. By (42),

$$\varphi(\mathsf{const}(k))\downarrow \in \Omega_{\overline{\alpha}}. \tag{43}$$

(3°) Compute $\varphi(\mathsf{const}(k))\downarrow e'$. By (43), $e' \in \Omega_\alpha$ and

$$f(\alpha(k)) \ =\ f(\overline{\alpha}(\mathsf{const}(k))) \ =\ \overline{\alpha}(\varphi(\mathsf{const}(k))) \ =\ \overline{\alpha}(e').$$

Hence by (39), (40) and (41),

$$\mathsf{d}\big(f(x),\ \overline{\alpha}(e')\big) \ = \ \mathsf{d}\big(f(x),\ f(\alpha(k))\big) \ < \ 2^{-n-1}. \tag{44}$$

(4°) Finally compute

$$y \ := \ \alpha\big(\{e'\}(n+1)\big). \tag{45}$$

This is possible by assumption (ii). Then, since $\alpha \circ \{e'\}$ is a fast Cauchy sequence,

$$\mathsf{d}\big(y,\ \overline{\alpha}(e')\big) \ = \ \mathsf{d}\big(\alpha(\{e'\}(n+1)),\ \overline{\alpha}(e')\big) \ \leq \ 2^{-n-1}. \tag{46}$$

Hence by (46) and (44),

$$\begin{aligned} \mathsf{d}\big(y,\ f(x)\big) \ &\leq \ \mathsf{d}\big(y, \overline{\alpha}(e')\big) \ + \ \mathsf{d}\big(\overline{\alpha}(e'),\ f(x)\big) \\ &< \ 2^{-n-1} \ + \ 2^{-n-1} \\ &= \ 2^{-n}. \end{aligned}$$

Now the value of y computed in (45) is the output of the algorithm for g. Note however that this value depends on the actual implementation of the 'choose' construct as used in the above algorithm. Therefore (in accordance with our semantics for the abstract model) we define $g_n(x)$ to be the set of all such y, for all possible implementations of 'choose'. Then g satisfies (38), and is $\boldsymbol{WhileCC}^*$ computable, by the above discussion.

9.3 $\boldsymbol{WhileCC}^*$-semicomputability of $\boldsymbol{dom}(f)$

Here we point out a connection between $\boldsymbol{WhileCC}^*$-semicomputability of the function domain and *strict* $\boldsymbol{WhileCC}^*$ approximability.

Definition 9.3.1. (a) The *halting set* of a $\boldsymbol{WhileCC}^*$ procedure $P\colon u \to v$ on A is

$$\{\, x \in A^u \mid P^A(x)\backslash\{\uparrow\} \neq \emptyset \,\}.$$

(b) A subset of A^u is $\boldsymbol{WhileCC}^*$*-semicomputable* if it is the halting set of some $\boldsymbol{WhileCC}^*$ procedure.

The following two lemmas have easy proofs.

LEMMA 9.3.2. *If U has a $\boldsymbol{WhileCC}^*$-effective open exhaustion, then U is $\boldsymbol{WhileCC}^*$-semicomputable. In fact, it is the halting set of P_{loc} (in the notation of Definition 9.1.3).*

LEMMA 9.3.3. *Suppose $\boldsymbol{dom}(f)$ is $\boldsymbol{WhileCC}^*$-semicomputable. Then*

f is $\boldsymbol{WhileCC}^*$-approximable $\iff$ f is strictly $\boldsymbol{WhileCC}^*$-approximable.

(Recall Definition 4.5.1.) Hence we see, by Lemmas 9.3.2 and 9.3.3, that the conclusion of Theorem B can be replaced by the (apparently) stronger statement:

If f is $\overline{\alpha}$-computable on A, then f is strictly $\boldsymbol{WhileCC}^*$ approximable on A.

10. COMPLETENESS OF $\boldsymbol{WhileCC}^*$ APPROXIMATION

Under certain assumptions, we can combine Theorems A and B into a single equivalence, namely Theorem C below. We will then look at several examples of metric algebras where our abstract and concrete models are equivalent according to this Theorem.

10.1 Completeness

We are ready to state the completeness theorem for $\boldsymbol{WhileCC}^*$ approximability relative to $\overline{\alpha}$-computability.

THEOREM C (COMPLETENESS). *Let A be an N-standard metric Σ-algebra, and (X, α) an enumerated $\boldsymbol{Sort}(\Sigma)$-subspace. Suppose the enumerated $\boldsymbol{Sort}(\Sigma)$-space $(\mathsf{C}_\alpha(X), \overline{\alpha})$ of α-computable elements of A is a Σ-subalgebra of A. Assume also that for all Σ-sorts s,*

(i) $\overline{\alpha}$ is strictly Σ-effective,

(ii) X_s is dense in A_s, and

(iii) $\alpha_s : \mathbb{N} \to A_s$ is $\boldsymbol{WhileCC}^*$-computable on A.

Let $f : A^u \rightharpoonup A_s$ be a function on A and V an open exhaustion of $\boldsymbol{dom}(f)$ such that

(iv) V is $\boldsymbol{WhileCC}^*$-effective, and

(v) f is effectively locally uniformly continuous w.r.t. V.

Then

$$f \text{ is } \boldsymbol{WhileCC}^* \text{ approximable on } A \iff f \text{ is } \overline{\alpha}\text{-computable on } A.$$

PROOF. From Theorems A and B. □

10.2 $\overline{\alpha}$-semicomputability of $\boldsymbol{dom}(f)$

(Compare §9.3.)

Definition 10.2.1 *($\overline{\alpha}$-semicomputability).* A subset of A^u is *$\overline{\alpha}$-semicomputable* if it is the domain of a strictly $\overline{\alpha}$-computable function.

LEMMA 10.2.2. *If $U \subseteq A^u$ is $\overline{\alpha}$-semicomputable then*

$$\overline{\alpha}^{-1}[U] \;=\; \{e \in \Omega^u_{\overline{\alpha}} \mid \overline{\alpha}(e) \downarrow \in U\} \;=\; S \cap \Omega^u_{\overline{\alpha}}$$

for some recursively enumerable set $S \subseteq \mathbb{N}$.

Remark 10.2.3. If $\overline{\alpha}^u$ is onto A^u, then the reverse implication of Lemma 10.2.2 holds.

LEMMA 10.2.4. *Let $f : A^u \rightharpoonup A_s$. Suppose $\boldsymbol{dom}(f)$ is $\overline{\alpha}$-semicomputable. Then*

$$f \text{ is } \overline{\alpha}\text{-computable} \iff f \text{ is strictly } \overline{\alpha}\text{-computable.}$$

PROOF. ($\Rightarrow$) Since $\boldsymbol{dom}(f)$ is $\overline{\alpha}$-semicomputable, by Lemma 10.2.2 there is an r.e. set S such that

$$\{e \in \Omega^u_{\overline{\alpha}} \mid \overline{\alpha}(e) \in \boldsymbol{dom}(f)\} \;=\; S \cap \Omega^u_{\overline{\alpha}}.$$

Now if φ is a computable $\overline{\alpha}$-tracking function for F, it can be replaced by a strict $\overline{\alpha}$-tracking function φ', defined by

$$\varphi'(e) \;\simeq\; \begin{cases} \varphi(e) & \text{if } \; e \in S \\ \uparrow & \text{otherwise} \end{cases}$$

which is easily seen to be computable. □

From this lemma and the discussion in §9.3, we have another form of the Theorem C, in which $\overline{\alpha}$-computability of $\boldsymbol{dom}(f)$ is (also) assumed (condition (vi) below).

THEOREM C^+. (COMPLETENESS FOR FUNCTIONS WITH SEMICOMPUTABLE DOMAIN) *Let A be an N-standard metric Σ-algebra, and (X,α) an enumerated $\boldsymbol{Sort}(\Sigma)$-subspace. Suppose the enumerated $\boldsymbol{Sort}(\Sigma)$-space $(\mathsf{C}_\alpha(X),\overline{\alpha})$ of α-computable elements of A is a Σ-subalgebra of A. Assume also that for all Σ-sorts s,*

(i) $\overline{\alpha}$ is strictly Σ-effective,

(ii) X_s is dense in A_s, and

(iii) $\alpha_s : \mathbb{N} \to A_s$ is $\boldsymbol{WhileCC}^*$-computable on A.

Let $f : A^u \rightharpoonup A_s$ be a function on A and V an open exhaustion of $\boldsymbol{dom}(f)$ such that

(iv) V is $\boldsymbol{WhileCC}^*$-effective in A,

(v) f is effectively locally uniformly continuous w.r.t. V, and

(vi) $\boldsymbol{dom}(f)$ is $\overline{\alpha}$-semicomputable.

Then

$$f \text{ is (strictly) } \boldsymbol{WhileCC}^* \text{ approximable on } A \iff f \text{ is (strictly) } \overline{\alpha}\text{-computable on } A.$$

Note that the word "strictly" may be omitted or inserted in either side at will.

10.3 Completeness for total effectively uniformly continuous functions

A special case of the Completeness Theorem, with a simpler formulation, is obtained by assuming that f is total and effectively globally uniformly continuous.

Note that since f is total, the difference between $\boldsymbol{WhileCC}^*$-approximability and strict $\boldsymbol{WhileCC}^*$-approximability, and between $\overline{\alpha}$-computability and strict $\overline{\alpha}$-computability, vanish, and applying Theorem C or C^+ (with the trivial exhaustion of A^u, which need not be mentioned explicitly) we obtain:

COROLLARY C^{tot}. (COMPLETENESS FOR TOTAL EFFECTIVELY UNIFORMLY CONTINUOUS FUNCTIONS) *Let A be an N-standard metric Σ-algebra, and (X,α) an enumerated $\boldsymbol{Sort}(\Sigma)$-subspace. Suppose the enumerated $\boldsymbol{Sort}(\Sigma)$-space $(\mathsf{C}_\alpha(X),\overline{\alpha})$ of α-computable elements of A is a Σ-subalgebra of A. Assume also that for all Σ-sorts s,*

(i) $\overline{\alpha}$ is strictly Σ-effective,

(ii) X_s is dense in A_s, and

(iii) $\alpha_s : \mathbb{N} \to A_s$ is $\boldsymbol{WhileCC}^*$-computable on A.

Let $f : A^u \to A_s$ be a total function on A such that

(iv) f is effectively uniformly continuous.

Then

$$f \text{ is } \boldsymbol{WhileCC}^* \text{ approximable on } A \iff f \text{ is } \overline{\alpha}\text{-computable on } A.$$

10.4 Examples of the application of the Completeness Theorem

(*a*) Canonical enumerations. The purpose of this example is to make plausible condition (*iii*) of Theorem C (and condition (*ii*) of Theorem B in Section 9), *i.e.*, the assumption of ***WhileCC**** computability of the enumeration α, by describing a commonly occurring situation which implies it.

Suppose (X, α) is an enumerated Σ-subalgebra of A.

Definition 10.4.1. The enumeration $\alpha : \mathbb{N} \twoheadrightarrow X$ is *effectively determined by a system of generators* $G = \langle g_0^s, g_1^s, g_2^s, \ldots \rangle_{s \in \boldsymbol{Sort}(\Sigma)}$ if, and only if,

(*i*) G generates X as a Σ-subalgebra of A;

(*ii*) α is defined as the composition of the maps

$$\mathbb{N} \xrightarrow{\boldsymbol{enum}_\Sigma} \boldsymbol{Term}(\Sigma) \xrightarrow{\boldsymbol{eval}_G} X$$

where $\boldsymbol{enum}_\Sigma$ is the inverse of the Gödel numbering of $\boldsymbol{Term}(\Sigma)$, and $\boldsymbol{eval}_G$ is the term evaluation induced by the mapping $\mathtt{x}_i^s \mapsto g_i^s$, $(i = 0, 1, 2, \ldots)$ for some standard enumeration $\mathtt{x}_0^s, \mathtt{x}_1^s, \mathtt{x}_2^s, \ldots$ of the Σ-variables of sort s; and

(*iii*) if, for any Σ-sort s, the sequence $\langle g_0^s, g_1^s, g_2^s, \ldots \rangle$ is finite, then each g_i^s is a Σ-constant, whereas if this sequence is infinite, then the map $i \mapsto g_i^s$ is a Σ-function.

An enumeration constructed in this way is called *canonical* w.r.t. G.

Remark 10.4.2 *(Totality of* $\boldsymbol{eval}_G$*).* We assume here that $\boldsymbol{eval}_G$ (and hence α) is total. This is achieved by assuming that either (*i*) A is total, or (*ii*) $\boldsymbol{Term}(\Sigma)$ is replaced by some decidable subset $\boldsymbol{Term}'(\Sigma)$ on which $\boldsymbol{eval}_G$ is total (for example, omitting all terms involving division by 0).

Either one of these assumptions holds in each of the following examples; for example, (*i*) holds in example (*b*) below, and (*ii*) in example (*c*), resulting in the same "canonical" enumeration α of $\mathbb{Q}$ in both cases (even though the algebras are different).

LEMMA 10.4.3. *If* α *is effectively determined by a system of generators, then the canonical enumerations* α_s *are* ***While**** *computable for all* Σ*-sorts* s*.*

PROOF. This follows from ***While**** computability of term evaluation [Tucker and Zucker 2000, Cor. 4.7]. □

The significance of the above definition and proposition is this: it is quite common for an enumeration to be effectively determined by a system of generators; and in such a situation, condition (*ii*) in Theorem B, and (*iii*) in Theorem C, will be (more than) satisfied. This will be the case in the following examples.

(*b*) Partial real algebra. Recall (Example (8.1.2) the enumeration α of $\mathbb{Q}$ as a subspace of the N-standardised metric algebra $\mathcal{R}_p^N$ of reals (Examples 3.5.4(*b*) and 3.6.1) and the corresponding enumeration $\overline{\alpha}$ of the set $\mathsf{C}_\alpha(\mathbb{Q})$ of *recursive reals*. Note that α is canonical, being effectively determined by the generators $\{0, 1\}$, and is hence ***While**** computable over $\mathcal{R}$. Further, $\mathbb{Q}$ is dense in $\mathbb{R}$, $\mathsf{C}_\alpha(\mathbb{Q})$ is a subfield of $\mathbb{R}$, and $\overline{\alpha}$ is strictly $\Sigma(\mathcal{R})$-effective. We then have, as another corollary to Theorem C^+:

COROLLARY 10.4.4. *Suppose* $f : \mathbb{R}^n \to \mathbb{R}$ *is effectively locally uniformly continuous w.r.t. some* ***WhileCC*****-effective open exhaustion of* $\mathbf{dom}(f)$*, and suppose* $\mathbf{dom}(f)$ *is* $\overline{\alpha}$*-semicomputable. Then*

$$f \text{ is (strictly) } \mathbf{WhileCC}^*\text{-approximable on } \mathcal{R}_p^N \iff f \text{ is (strictly) } \overline{\alpha}\text{-computable on } \mathbb{R}.$$

(Here, again, the word "strictly" may be omitted or inserted in either side at will).

Examples of functions satisfying the assumption (and also the equivalence) are all the common (partial) functions of elementary calculus, such as $1/x$, $\log x$ and $\tan x$.

Consider the special case of total functions on the *unit interval* $I = [0, 1]$. (Recall that a continuous function on I is uniformly continuous, so we may as well assume effective global uniform continuity on I.) Applying Corollary $\mathrm{C}^{\mathrm{tot}}$ to the partial interval algebra $\mathcal{I}_p^N$ (Example 3.5.4(c)) and a canonical enumeration α of $\mathbb{Q} \cap I$, we obtain:

COROLLARY 10.4.5. *Suppose* $f : I^n \to I$ *is effectively uniformly continuous. Then*

$$f \text{ is } \mathbf{WhileCC}^*\text{-approximable on } \mathcal{I}_p^N \iff f \text{ is } \overline{\alpha}\text{-computable on } I.$$

(*c*) Banach spaces with countable bases. Let X be a Banach space over $\mathbb{R}$ with a countable basis $e_0, e_1, e_2, \ldots$, which means that any element $x \in X$ can be represented uniquely as an infinite sum

$$x = \sum_{i=0}^{\infty} r_i e_i$$

with coefficients $r_i \in \mathbb{R}$ (where the infinite sum is understood as denoting convergence of the partial sums in the norm of X). (Background on Banach space theory can be found in any of the standard texts, *e.g.*, [Royden 1963; Taylor and Lay 1980].) To program with X, we construct a many-sorted algebra $\mathcal{X}$:

algebra	$\mathcal{X}$
import	$\mathcal{R}_p^N$
carriers	X
functions	$0\colon \to X$,
	$+\colon X^2 \to X$,
	$-\colon X \to X$,
	$\odot\colon \mathbb{R} \times X \to X$,
	$\|\cdot\|\colon X \to \mathbb{R}$,
	$\mathsf{e}\colon \mathbb{N} \to X$,
	$\mathsf{if}_X\colon \mathbb{B} \times X^2 \to X$
end	

where $\odot$ is scalar multiplication, $\|\cdot\|$ is the norm function and and e is the enumeration of the basis: $\mathsf{e}(i) = e_i$. Note that the algebras $\mathcal{B}$ and $\mathcal{N}$ are implicitly imported, as parts of $\mathcal{R}_p^N$, so that there are four carriers: X, $\mathbb{R}$, $\mathbb{B}$ and $\mathbb{N}$, of sorts vector, scalar, bool and nat respectively.

Let $\Sigma = \Sigma(\mathcal{X})$. Let Σ_0 be Σ without the norm function $\|\cdot\|$, and let $\mathcal{X}_0$ be the reduct [2] of $\mathcal{X}$ to Σ_0. Then Σ_0 is the signature of an N-standardised vector space over $\mathbb{R}$, with explicit countable basis.

This can be turned into a metric algebra in the standard way, by defining a distance function on X in terms of the norm $\mathsf{d}(x,y) =_{df} \|x-y\|$.

Let $\mathrm{L}(\mathbb{Q},\mathsf{e}) \subset X$ be the set of all finite linear combinations of basis elements from e with coefficients in $\mathbb{Q}$. The following are easily shown:

—$\mathrm{L}(\mathbb{Q},\mathsf{e})$ is countable; in fact it has a canonical enumeration $\alpha : \mathbb{N} \twoheadrightarrow \mathrm{L}(\mathbb{Q},\mathsf{e})$ w.r.t. the generators e, which (by (a) above) is ***While**** computable;

—$\mathrm{L}(\mathbb{Q},\mathsf{e})$ is dense in X;

—$\mathrm{L}(\mathbb{Q},\mathsf{e})$, with scalar field $\mathbb{Q}$ (with carriers $\mathbb{N}$ and $\mathbb{B}$) is a Σ_0-subalgebra of $\mathcal{X}_0$.

Now let $(\mathsf{C}_\alpha(\mathrm{L}(\mathbb{Q},\mathsf{e})), \overline{\alpha})$ be the enumerated subspace of α-computable vectors. Then

—$\mathsf{C}_\alpha(\mathrm{L}(\mathbb{Q},\mathsf{e}))$, with scalar field $\mathsf{C}_\alpha(\mathbb{Q})$ (with carriers $\mathbb{N}$ and $\mathbb{B}$) is also a Σ_0-subalgebra of $\mathcal{X}_0$; and moreover,

—$\overline{\alpha}$ is strictly Σ_0-effective.

However $\mathsf{C}_\alpha(\mathrm{L}(\mathbb{Q},\mathsf{e}))$ is *not* necessarily a *normed* subspace of $\mathcal{X}$, since it may not be closed under $\|\cdot\|$, *i.e.*, $\|x\|$ may not be in $\mathsf{C}_\alpha(\mathbb{Q})$ for all $x \in \mathsf{C}_\alpha(\mathrm{L}(\mathbb{Q},\mathsf{e}))$; for example, if $\mathcal{X}$ is the space ℓ^p or $L^p[0,1]$ where p is a nonrecursive real (see Examples 10.4.9 below). We must therefore make an explicit assumption for the Banach space $(X, \|\cdot\|)$ with respect to both the *closure* of $\mathsf{C}_\alpha(\mathrm{L}(\mathbb{Q},\mathsf{e}))$ under $\|\cdot\|$, and the $\overline{\alpha}$-*computability* of $\|\cdot\|$.

ASSUMPTION 10.4.6 ($\overline{\alpha}$-COMPUTABLE NORM ASSUMPTION FOR $(X, \|\cdot\|)$). *For all $x \in \mathsf{C}_\alpha(L(\mathbb{Q},\mathsf{e}))$, $\|x\| \in \mathsf{C}_\alpha(\mathbb{Q})$. Further, the norm function $\|\cdot\|$ is $\overline{\alpha}$-computable.*

As we will see, many common examples of Banach spaces satisfy this assumption.

Note that Assumption 10.4.6 is equivalent to the following (apparently weaker) assumption, which is often easier to prove:

ASSUMPTION 10.4.7 ($(\alpha,\overline{\alpha})$-COMPUTABLE NORM ASSUMPTION FOR $(X, \|\cdot\|)$). *For all $x \in L(\mathbb{Q},\mathsf{e})$, $\|x\| \in \mathsf{C}_\alpha(\mathbb{Q})$. Further, $\|\cdot\|$ has a computable $(\alpha,\overline{\alpha})$-tracking function, i.e., a computable function $\varphi : \mathbb{N} \to \mathbb{N}$ such that the following diagram commutes:*

$$\begin{array}{ccc} L(\mathbb{Q},\mathsf{e}) & \xrightarrow{\|\cdot\|} & \mathsf{C}_\alpha(\mathbb{Q}) \\ {\scriptstyle\alpha}\uparrow & & \uparrow{\scriptstyle\overline{\alpha}} \\ \mathbb{N} & \xrightarrow[\varphi]{} & \Omega_{\overline{\alpha}} \end{array}$$

Suppose now that $(X, \|\cdot\|)$ satisfies the $\overline{\alpha}$-computable norm assumption. Then the Σ_0-subalgebra $\mathsf{C}_\alpha(\mathrm{L}(\mathbb{Q},\mathsf{e}))$ of $\mathcal{X}_0$ can be expanded to a Σ-subalgebra of $\mathcal{X}$ (which we will also write as $\mathsf{C}_\alpha(\mathrm{L}(\mathbb{Q},\mathsf{e}))$), enumerated by $\overline{\alpha}$, which is strictly Σ-effective.

[2] Reducts of algebras are defined in [Tucker and Zucker 2000, Def. 2.6].

Now let $F: X \to \mathbb{R}$ be a (total) *linear functional on* X. F is said to be *bounded* if for some real M,

$$|F(x)| \;\leq\; M\|x\| \quad \text{for all } x \in X. \tag{47}$$

Write $\|F\|$ for the least M for which (47) holds. Then if F is bounded,

$$|F(x) - F(y)| \;\leq\; \|F\| \cdot \|x - y\| \quad \text{for all } x, y \in X,$$

and so F is uniformly continuous, in fact it is clearly *effectively uniformly continuous*. We may therefore apply Corollary $\mathrm{C}^{\mathrm{tot}}$ to F.

COROLLARY 10.4.8. *Let X be a Banach space over $\mathbb{R}$ with countable basis, and let $\mathsf{C}_\alpha(L(\mathbb{Q}, \mathsf{e}))$ be the enumerated subspace of α-computable vectors, where α is a canonical enumeration of the subspace $L(\mathbb{Q}, \mathsf{e})$. Suppose $(X, \|\cdot\|)$ satisfies the $(\alpha, \overline{\alpha})$-computable norm assumption. Then for any bounded linear functional F on X,*

$$F \text{ is } \textbf{\textit{WhileCC}}^* \text{ approximable on } \mathcal{X} \iff F \text{ is } \overline{\alpha}\text{-computable on } X.$$

Here $\mathcal{X}$ is the N-standard algebra formed from X as above.

Finally we give examples of Banach spaces which satisfy the $\overline{\alpha}$-computable norm assumption.

Examples 10.4.9 *(Banach spaces with computable norms).*
(*a*) For $1 \leq p < \infty$, we have the space ℓ^p of all sequences $x = \langle x_n \rangle_{n=0}^{\infty}$ of reals such that $\sum_{n=0}^{\infty} |x_n|^p < \infty$, with norm defined by

$$\|x\|_p \;=\; \Big(\sum_{n=0}^{\infty} |x_n|^p\Big)^{1/p},$$

and a countable basis given by $e_i = \langle e_{i,n} \rangle_{n=0}^{\infty}$, where

$$e_{i,n} \;=\; \begin{cases} 1 & \text{if } i = n, \\ 0 & \text{otherwise.} \end{cases}$$

It is easy to see that

if p is a recursive real, then ℓ^p satisfies the computable norm assumption,

and so Corollary 10.4.8 can be applied to it.

(*b*) For $1 \leq p < \infty$, we have the space $L^p[0,1]$ of all Lebesgue measurable functions $x(t)$ on the unit interval $[0,1]$ such that $\int_0^1 |x|^p < \infty$, with norm defined by

$$\|x\|_p \;=\; \Big(\int_0^1 |x|^p\Big)^{1/p},$$

and a countable basis given by (*e.g.*) some standard enumeration of all step functions on $[0,1]$ with rational values and (finitely many) rational points of discontinuity, or of all polynomial functions on $[0,1]$ with rational coefficients. Again, we see that

if p is a recursive real, then $L^p[0,1]$ satisfies the computable norm assumption.

(*c*) The space $C[0,1]$ of all continuous functions $x(t)$ on $[0,1]$, with norm

$$\|x\|_{\sup} = \sup_{t \in I} |x(t)|$$

and a countable basis given by a standard enumeration of all zig-zag functions on $[0,1]$ with (finitely many) turning points with rational coordinates, or of all polynomial functions on $[0,1]$ with rational coefficients. Again,

$C[0,1]$ *satisfies the computable norm assumption.*

11. CONCLUSION

We have compared two theories of computable functions on topological algebras, one based on an abstract, high level model of programming and another based on a concrete, low-level implementation model. Our examples and results here, combined with our earlier results [Tucker and Zucker 1999; 2000] and those of Brattka [Brattka 1996; 1999], show that the following are surprisingly necessary features of a comprehensive theory of computation on topological algebras:

1. The algebras have partial operations.
2. Functions are both continuous and many-valued.
3. Classical algorithms in analysis require nondeterministic constructs for their proper expression in programming languages.
4. Indeed, many-valued subfunctions are needed to compute even single-valued functions, and abstract models must be nondeterministic even to compute deterministic problems.
5. Abstract models and effective approximations by abstract models are generally sound for concrete models.
6. Abstract models even with approximation or limit operators are adequate to capture concrete models only in special circumstances.
7. Nevertheless there are interesting examples where equivalence holds.
8. The classical computable functions of analysis can be characterised by abstract models of computation.

Specifically, we examined abstract computation by the basic imperative model of 'while'-array programs. Many algorithms in practical computation are presented in pseudo-code based on the 'while' language. To meet the requirement of feature 2 above we added the simplest form of countable choice to the assignments of the language, and we defined the $\boldsymbol{WhileCC}^*$ approximable computations. We proved a Soundness Theorem (Theorem A) and an Adequacy Theorem (Theorem B), and combined these into a Completeness Theorem (Theorem C), in the case of metric algebras with partial operations. We considered algebras of real numbers and Banach spaces where equivalence theorems hold.

Interesting technical questions arise in working out the details of the computability theory for the $\boldsymbol{WhileCC}^*$ model (*cf.* the theory for single-valued functions on total algebras in [Tucker and Zucker 2000]). Other important abstract models of computation, for example the schemes in [Brattka 1999], could be extended with nondeterminsitic constructs in order to establish equivalence with concrete models. The topological properties of many-valued functions also need investigation.

Returning to the general problem posed in the Introduction, the features 1–8 above suggest that new research directions are needed to develop a comprehensive theory of specification, computation and reasoning with infinite data. What are the appropriate programming constructs for topological computations? What specification techniques are appropriate for continuous systems? What logics are needed to support verification of programs that approximate functions? Our work on computation suggests that some advanced semantic features are needed; in particular, the nondeterminism that was important in programming methodologies of the late 1970s (*e.g.*, [Dijkstra 1976]) seems necessary for the proper development of topological programming. There are plenty of algorithms in scientific modelling, numerical analysis and graphics to investigate, using such new theoretical tools.

The paper [Tucker and Zucker 2002b] is a sequel to this paper.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library.

ACKNOWLEDGMENTS

We thank Vasco Brattka and Kristian Stewart for valuable discussions, and three anonymous referees for helpful comments.

REFERENCES

ABERTH, O. 1980. *Computable Analysis.* MIT Press.

ABERTH, O. 2001. *Computable Calculus.* Addison-Wesley.

APT, K. AND OLDEROG, E.-R. 1991. *Verification of Sequential and Concurrent Programs.* Springer-Verlag.

BLUM, L., CUCKER, F., SHUB, M., AND SMALE, S. 1998. *Complexity and Real Computation.* Springer-Verlag.

BLUM, L., SHUB, M., AND SMALE, S. 1989. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin of the American Mathematical Society 21*, 1–46.

BRATTKA, V. 1996. Recursive characterisation of computable real-valued functions and relations. *Theoretical Computer Science 162*, 45–77.

BRATTKA, V. 1999. Recursive and computable operations over topological structures. Ph.D. thesis, FernUniversität Hagen, Fachbereich Informatik, Hagen, Germany. Informatik Berichte 255, FernUniversität Hagen, July 1999.

CEITIN, G. 1959. Algebraic operators in constructive complete separable metric spaces. *Doklady Akademii Nauk SSSR 128*, 49–52.

DE BAKKER, J. 1980. *Mathematical Theory of Program Correctness.* Prentice Hall.

DIJKSTRA, E. 1976. *A Discipline of Programming.* Prentice Hall.

EDALAT, A. 1995. Dynamical systems, measures, and fractals via domain theory. *Information and Computation 120*, 32–48.

EDALAT, A. 1997. Domains for computation in mathematics, physics and exact real arithmetic. *Bulletin of Symbolic Logic 3*, 401–452.

ENGELKING, R. 1989. *General Topology.* Heldermann Verlag.

GOGUEN, J., THATCHER, J., AND WAGNER, E. 1978. An initial approach to the specification, correctness and implementation of abstract data types. In *Current Trends in Programming Methodology, vol. 4: Data Structuring*, R. Yeh, Ed. Prentice Hall, 80–149.

GRZEGORCZYK, A. 1955. Computable functions. *Fundamenta Mathematicae 42*, 168–202.

GRZEGORCZYK, A. 1957. On the defintions of computable real continuous functions. *Fundamenta Mathematicae 44*, 61–71.

HEATH, M. 1997. *Scientific Computing: An Introductory Survey.* McGraw-Hill.

LACOMBE, D. 1955. *Extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles*, I, II, III. *C.R. Acad. Sci. Paris.* 240:2470–2480, 241:13–14,151–153.

MESEGUER, J. AND GOGUEN, J. 1985. Initiality, induction and computability. In *Algebraic Methods in Semantics*, M. Nivat and J. Reynolds, Eds. Cambridge University Press, 459–541.

MOSCHOVAKIS, Y. 1964. Recursive metric spaces. *Fundamenta Mathematicae 55*, 215–238.

ODIFREDDI, P. 1999. *Classical Recursion Theory* (21st ed.). North Holland.

POUR-EL, M. AND RICHARDS, J. 1989. *Computability in Analysis and Physics.* Springer-Verlag.

ROYDEN, H. 1963. *Real Analysis.* Macmillan.

SPREEN, D. 1998. On effective topological spaces. *Journal of Symbolic Logic 63*, 185–221.

SPREEN, D. 2001. Representations versus numberings: on the relationship of two computability notions. *Theoretical Computer Science 263*, 473–499.

STEPHENSON, K. 1996. An algebraic approach to syntax, semantics and computation. Ph.D. thesis, Department of Computer Science, University of Wales, Swansea.

STEWART, K. 1998. Concrete and abstract models of computation over metric algebras. Ph.D. thesis, Department of Computer Science, University of Wales, Swansea.

STOLTENBERG-HANSEN, V. AND TUCKER, J. 1988. Complete local rings as domains. *Journal of Symbolic Logic 53*, 603–624.

STOLTENBERG-HANSEN, V. AND TUCKER, J. 1995. Effective algebras. In *Handbook of Logic in Computer Science*, S. Abramsky, D. Gabbay, and T. Maibaum, Eds. Vol. 4. Oxford University Press, 357–526.

STOLTENBERG-HANSEN, V. AND TUCKER, J. 1999. Concrete models of computation for topological algebras. *Theoretical Computer Science 219*, 347–378.

TAYLOR, A. AND LAY, D. 1980. *Introduction to Functional Analysis.* John Wiley & Sons.

TUCKER, J. AND ZUCKER, J. 1988. *Program Correctness over Abstract Data Types, with Error-State Semantics.* CWI Monographs, vol. 6. North Holland.

TUCKER, J. AND ZUCKER, J. 1992a. Examples of semicomputable sets of real and complex numbers. In *Constructivity in Computer Science: Summer Symposium, San Antonio, Texas, June 1991*, J. Myers, Jr. and M. O'Donnell, Eds. Lecture Notes in Computer Science, vol. 613. Springer-Verlag, 179–198.

TUCKER, J. AND ZUCKER, J. 1992b. Theory of computation over stream algebras, and its applications. In *Mathematical Foundations of Computer Science 1992: 17th International Symposium, Prague*, I. Havel and V. Koubek, Eds. Lecture Notes in Computer Science, vol. 629. Springer-Verlag, 62–80.

TUCKER, J. AND ZUCKER, J. 1994. Computable functions on stream algebras. In *Proof and Computation: NATO Advanced Study Institute International Summer School at Marktoberdorf, 1993*, H. Schwichtenberg, Ed. Springer-Verlag, 341–382.

TUCKER, J. AND ZUCKER, J. 1999. Computation by 'while' programs on topological partial algebras. *Theoretical Computer Science 219*, 379–420.

TUCKER, J. AND ZUCKER, J. 2000. Computable functions and semicomputable sets on many-sorted algebras. In *Handbook of Logic in Computer Science*, S. Abramsky, D. Gabbay, and T. Maibaum, Eds. Vol. 5. Oxford University Press, 317–523.

TUCKER, J. AND ZUCKER, J. 2002a. Abstract computability and algebraic specification. *ACM Transactions on Computational Logic 3*, 279–333.

TUCKER, J. AND ZUCKER, J. 2002b. Computable total functions, algebraic specifications and dynamical systems. Tech. Rep. CAS 02-04-JZ, Department of Computing & Software, McMaster University. October. Submitted.

WEIHRAUCH, K. 2000. *Computable Analysis: An Introduction.* Springer-Verlag.

WIRSING, M. 1991. Algebraic specification. In *Handbook of Theoretical Computer Science, Vol. B: Formal Methods and Semantics*, J. van Leeuwen, Ed. North Holland, 675–788.

Received October 2001; revised April 2003; accepted April 2003