

# Characterizations of semicomputable sets of real numbers<sup>☆</sup>



Bo Xie<sup>1</sup>, Ming Quan Fu, Jeffery Zucker\*

Department of Computing and Software, McMaster University, Hamilton, Ontario L8S 4K1, Canada

## ARTICLE INFO

### Article history:

Available online 20 November 2013

### Keywords:

Computability on reals  
Computability on topological algebras  
Engeler's Lemma  
Semicomputable sets of reals

## ABSTRACT

We give some characterizations of semicomputability of sets of reals by programs in certain **While** programming languages over a topological partial algebra of reals. We show that such sets are semicomputable if and only if they are one of the following:

- (i) unions of effective sequences of disjoint algebraic open intervals;
- (ii) unions of effective sequences of rational open intervals;
- (iii) unions of effective sequences of algebraic open intervals.

For the equivalence (i), the **While** language must be augmented by a strong OR operator, and for equivalences (ii) and (iii) it must be further augmented by a strong existential quantifier over the naturals (**While**<sup>∃N</sup>). We also show that the class of **While**<sup>∃N</sup> semicomputable relations on reals is closed under projection. The proof makes essential use of the continuity of the operations of the algebra.

© 2013 Elsevier Inc. All rights reserved.

## 1. Introduction

### 1.1. Background

Our research in this paper is based on computations by high level programming languages featuring the ‘while’ construct over many-sorted topological partial algebras.

An algebra  $A$  is a finite family of sets

$$A_{s_1}, \dots, A_{s_n}$$

called *carriers* of sorts  $s_1, \dots, s_n$ , and a finite set of (total or partial) functions defined over these sets.<sup>2</sup>

An algebra is said to be *standard* if it contains the sort of booleans and the standard boolean operators. It is  $N$ -standard if in addition, it contains the sort of naturals and the standard arithmetic operations.

Classical computability theory on naturals has been studied since the 1930's. There are many extensions of this theory to abstract structures. One of these extensions has been the investigation of total (non-topological) algebras of reals [1]. A detailed discussion of such extensions is given in [14]. We have adapted many of the definitions and proofs from [14] to fit topological partial algebras.

There are two kinds of computational models for algebras: *abstract* and *concrete*. Abstract models are independent of the representations of the data types of the algebras, while concrete models are dependent on such representations. Typically

<sup>☆</sup> Research supported by a grant from the Natural Sciences and Engineering Research Council (Canada).

\* Corresponding author.

E-mail addresses: tylerxie@yahoo.com (B. Xie), fumq@mcmaster.ca (M.Q. Fu), zucker@mcmaster.ca (J. Zucker).

<sup>1</sup> Current address: Watermark Insurance Services Inc., 1020 Brock Road South, Suite 2005, Pickering, Ontario L1W 3H2, Canada.

<sup>2</sup> We consider constants to be 0-ary functions.

abstract models are based on high level programming language, such as the **While** language. This is an imperative programming language with the basic operations of assignment, sequential composition, conditional and the ‘while’ loop.

Examples of concrete models over  $\mathbb{R}$  are the classical computable analysis of Pour-El and Richards [7], and TTE (Type-2 Theory of Effectivity) of Weihrauch [20]; both these models represent reals as effective Cauchy sequences of rationals, and their equivalence follows from the results in [11].

Some work in bridging the gap between abstract and concrete models is made in [15,16]. We will discuss this issue again in Section 6.2.

In studying computability theory on abstract algebras, we take, as a guiding principle, the *Continuity Principle* [13,15]:

$$\text{computability} \implies \text{continuity}.$$

(This principle is ignored in [1].)

We will focus on the N-standard topological partial algebra  $\mathcal{R}$ , which is formed from the “N-standardization” of the ring of reals, by adding the two boolean-valued partial operations:

$$\text{eq}_{\mathbb{R}}, \text{less}_{\mathbb{R}} : \mathbb{R}^2 \rightarrow \mathbb{B}.$$

It follows from the Continuity Principle that these operations have to be partial. (This is because the set of reals is connected and the booleans are discrete, so the only total continuous functions from the reals to the booleans are constants.)

Abstract models of computability such as the **While** language, with partial basic operations on  $\mathbb{R}$ , suffer from a limitation, namely the inability to implement *interleaving* or *dovetailing*. The problem is that when interleaving two processes, one process may converge and the other diverge locally (because of the partiality of the basic operations). The resulting process will then diverge, whereas we would want it to converge.

To correct this deficiency, we establish two enhancements of the **While** language and construct two new languages: **While**<sup>OR</sup> and **While**<sup>∃N</sup>.

In the **While**<sup>OR</sup> language, we introduce a strong disjunction operation ‘∇’, where  $b_1 \nabla b_2$  converges to true if either component converges to true, even if the other one diverges. By means of this, interleaving of finitely many processes can be simulated at the abstract level.

The **While**<sup>∃N</sup> language includes a strong ‘Exist’ construct over the naturals:

$$x^{\mathbb{B}} := \exists z P(t, z)$$

where  $z$  is a nat variable and  $P$  is a boolean-valued procedure. By means of this, interleaving of *infinitely* many processes can be simulated at the abstract level.

We will study the structure of semicomputable sets of reals in  $\mathcal{R}$ , where a set is said to be (for example) **While** semicomputable if it is the halting set of a **While** procedure.

## 1.2. Results

We will prove certain structure theorems for semicomputable sets of reals in  $\mathcal{R}$ :

- (1) **While**<sup>OR</sup> semicomputable  $\iff$  union of an effective countable seq. of *disjoint algebraic* intervals.<sup>3,4</sup>
- (2) **While**<sup>∃N</sup> semicomputable  $\iff$  union of an effective countable sequence of *algebraic* intervals.
- (3) **While**<sup>∃N</sup> semicomputable  $\iff$  union of an effective countable sequence of *rational* intervals.<sup>5</sup>

We have no structure theorem for **While** semicomputability over  $\mathcal{R}$ , only a partial result:

- (4) (a) **While** semicomputable  $\implies$  countable union of eff. sequence of *rational* intervals;
- (b) **While** semicomputable  $\iff$  countable union of eff. sequence of *disjoint rational* intervals.

In (1) and (4), we need disjointedness because the **While** and **While**<sup>OR</sup> languages cannot implement interleaving of infinitely many processes over partial algebras. For that we need the ‘Exist’ construct, as in (2) and (3).

The main tools in proving these results are:

- (a) Engeler’s Lemma for standard topological partial algebras, which states (roughly) that a semicomputable set can be expressed as the disjunction of an effective infinite sequence of booleans. It is proved by constructing a computation tree for the procedure being considered.

<sup>3</sup> By “interval” we will always mean *open* interval of reals.

<sup>4</sup> An algebraic interval is an interval between two algebraic numbers.

<sup>5</sup> A rational interval is an interval between two rational numbers.

- (b) The Canonical Form Lemma for booleans over  $\mathcal{R}$ , which states that a boolean term over  $\mathcal{R}$  can be expressed as a boolean combination of polynomial equations and inequalities.
- (c) The Partition Lemma for booleans over  $\mathcal{R}$ , which states that a boolean term with only one real variable partitions the real line into finitely many disjoint “positive intervals”, “negative intervals”, and “points of divergence”. The proof is by structural induction on the boolean, using the Canonical Form Lemma.

Note that Engeler’s Lemma applies to all standard topological partial algebras, whereas the Canonical Form and Partition Lemmas apply only in special cases, such as the algebra  $\mathcal{R}$ .

The sequence of booleans given by Engeler’s Lemma for  $\mathbf{While}^{(\text{OR})}$  has a *semantic disjointedness* property, which is used in the ‘ $\implies$ ’ direction of the proof of (1). This property does not hold for  $\mathbf{While}^{\exists\mathbb{N}}$ , because of the special nature of the associated “computation hypertree”, which is not strictly a tree, but a directed acyclic graph.

### 1.3. Overview of the paper

Section 2 reviews some preliminaries on numerical codings, computable reals and basic algebraic results.

Section 3 defines the fundamental concepts of signature, algebra, standard and N-standard algebra, and topological partial algebra, and describes the topological partial algebra  $\mathcal{R}$ , which is used throughout the paper.

It also gives the syntax and semantics of the  $\mathbf{While}$ ,  $\mathbf{While}^{\text{OR}}$  and  $\mathbf{While}^{\exists\mathbb{N}}$  languages, and reviews the notions of computability, relative computability, semicomputability and projective semicomputability with respect to the  $\mathbf{While}$  language and its variants.

In Section 4 we prove Engeler’s Lemma for the  $\mathbf{While}$ ,  $\mathbf{While}^{\text{OR}}$  and  $\mathbf{While}^{\exists\mathbb{N}}$  languages over N-standard partial algebras. To prove this lemma, two kinds of computation trees are constructed, one for  $\mathbf{While}$  and  $\mathbf{While}^{\text{OR}}$ , and the other, a “hypertree”, for  $\mathbf{While}^{\exists\mathbb{N}}$ .

Section 5 focuses on the algebra  $\mathcal{R}$  of reals. It gives a “modified semantics” for atomic booleans in the language of  $\mathcal{R}$ . It then presents the Canonical Form and Partition Lemmas, followed by the four structure theorems listed above. This section concludes with a proof of the theorem that  $\mathbf{While}^{\exists\mathbb{N}}$  semicomputability on  $\mathcal{R}$  is closed under projection, i.e., a projection of a  $\mathbf{While}^{\exists\mathbb{N}}$  semicomputable set of reals is again  $\mathbf{While}^{\exists\mathbb{N}}$  semicomputable. This result is interesting because it does not hold over the total (non-topological) algebra over the reals studied in [14]. We do not know if it holds for  $\mathbf{While}$  or  $\mathbf{While}^{\text{OR}}$  over  $\mathcal{R}$ .

Section 6 contains some ideas for future work. The most interesting (and challenging) of these would be a generalization of the Partition Lemma, and (hence) the structure theorems, to more than one dimension.

## 2. Preliminaries

### 2.1. Numerical codings

We assume given families of effective numberings, i.e. surjective codings of the syntactic classes with which we deal, with  $\lceil E \rceil$  denoting the code of the expression  $E$ .

These numberings are standard, so we will assume that we can primitive recursively simulate all operations involved in processing the syntax of the programming language.

Further, we can define, in a standard way, numberings or codings of the sets  $\mathbb{N}^2$ ,  $\mathbb{N}^*$ ,  $\mathbb{Z}$  and  $\mathbb{Q}$ . We write  $\langle x, y \rangle$  for the code of a pair  $(x, y) \in \mathbb{N}^2$ ,  $[x_1, \dots, x_n]$  for the code of a tuple  $(x_1, \dots, x_n) \in \mathbb{N}^*$  ( $n \geq 0$ ), and more generally,  $\lceil x \rceil$  for the code of an element  $x$  of  $\mathbb{Z}$ ,  $\mathbb{Q}$ , etc.

By “effective(ly)”, we mean *effective* in the codes of the syntactic or mathematical objects referred to.

### 2.2. Computable reals

**Definition 2.2.1** (*Computable sequence of rationals*). A sequence  $(r_0, r_1, r_2, \dots)$  of rationals is *computable* if the function  $n \mapsto \lceil r_n \rceil$  is recursive. A *code* of the sequence can be defined as an index of this recursive function.

**Definition 2.2.2** (*Computable real number*). A real number  $x$  is *computable* if there exist

- (1) a computable sequence  $(r_n)$  of rationals converging to  $x$ , and
- (2) a computable modulus of convergence, i.e., a total recursive strictly increasing function  $M : \mathbb{N} \rightarrow \mathbb{N}$  such that  $\forall n$ ,  $|r_n - x| < 2^{-M(n)}$ .

A *code* of a computable real  $x$  is then defined as a pair  $\langle e, m \rangle$  where  $e$  is an index of a convergent sequence  $(r_n)$  for  $x$ , and  $m$  is an index for its modulus of convergence.

**Lemma 2.2.3.** For each (code for a) computable real number  $x$ , we can effectively construct computable sequences of rationals  $(r_n)$  and  $(s_n)$  such that  $(r_n)$  is increasing and  $(s_n)$  is decreasing and for all  $n$ :

$$0 < (x - r_n) < 2^{-n} \quad \text{and} \\ 0 < (s_n - x) < 2^{-n}.$$

**Proof.** The construction of the sequences  $(r_n)$  and  $(s_n)$  as required from a computable sequence for  $x$  is straightforward.  $\square$

By “polynomial” we will mean a polynomial with integer coefficients. A unary polynomial is a polynomial in one variable. An algebraic number is a root of a unary polynomial. The code of an algebraic number  $\alpha$  can be defined as  $\ulcorner \alpha \urcorner = \langle \ulcorner p \urcorner, k \rangle$ , where  $\alpha$  is the  $k$ -th smallest real root of the polynomial  $p$ , and  $p$  is a minimal polynomial for  $\alpha$ .

**Lemma 2.2.4.** Let  $\mathbb{A}$  and  $\mathbb{R}_c$  be the sets of algebraic and computable real numbers respectively. Then the embeddings

$$\mathbb{Q} \hookrightarrow \mathbb{A} \hookrightarrow \mathbb{R}_c$$

are effective in the respective codings.

In other words, there is a computable function  $f : \mathbb{N} \rightarrow \mathbb{N}$  such that if  $k$  is the code of a rational, then  $f(k)$  is the code of the same number viewed as an algebraic number. Similarly for  $\mathbb{A} \hookrightarrow \mathbb{R}_c$ .

On  $\mathbb{R}$ , the open intervals

$$(a, b), \quad (-\infty, a), \quad (b, \infty)$$

are called rational, algebraic or computable real intervals according as  $a$  and  $b$  are rational, algebraic or computable reals respectively.

We can give a coding for such intervals in an obvious way.

**Lemma 2.2.5.** Let  $(c, d)$  be a computable interval. We can effectively find a sequence of expanding rational intervals  $(r_i, s_i)$  such that

$$(c, d) = \bigcup_{i=0}^{\infty} (r_i, s_i).$$

**Proof.** By Lemma 2.2.3.  $\square$

### 2.3. Basic algebraic results

The following results can be found, with proofs, in standard texts on algebra [19,5] and real analysis [8,9].

**Proposition 2.3.1.** A non-zero unary polynomial of degree  $n$  has at most  $n$  real roots.

**Corollary 2.3.2.** If a polynomial  $p(x_1, \dots, x_m)$  has the value 0 at all points in  $\mathbb{R}^m$ , then it must be the zero polynomial.

**Proof.** By induction on  $m$ , using Proposition 2.3.1.  $\square$

**Proposition 2.3.3 (Intermediate value theorem).** Let  $f$  be a real-valued function that is continuous on the closed interval  $[a, b]$ . Suppose  $f(a)$  and  $f(b)$  have different signs. Then there exists  $c \in (a, b)$  such that  $f(c) = 0$ .

**Corollary 2.3.4.** A unary polynomial  $p$  of degree  $n > 0$  with  $m (\leq n)$  distinct real roots  $\alpha_1, \dots, \alpha_m$  defines  $m + 1$  algebraic intervals:

$$(-\infty, \alpha_1), (\alpha_1, \alpha_2), \dots, (\alpha_{m-1}, \alpha_m), (\alpha_m, \infty)$$

in each of which  $p$  is either only positive or only negative.

**Lemma 2.3.5.** Given any unary polynomial  $p$  of degree  $n$ , we can find, effectively in  $\ulcorner p \urcorner$ :

(1) the number of distinct real roots  $m (\leq n)$  of  $p$ , and, writing these as

$$\alpha_1 < \alpha_2 < \dots < \alpha_m,$$

- (2) – a rational less than  $\alpha_1$ ,  
 – a rational between  $\alpha_k$  and  $\alpha_{k+1}$ , for  $1 \leq k < m$ , and  
 – a rational bigger than  $\alpha_m$ .

**Proof.** From Sturm's theorem [19].  $\square$

### 3. While computation on standard partial algebras

We study a number of high level imperative programming languages based on the 'while' construct, applied to a many-sorted signature  $\Sigma$ . We give semantics for these languages relative to a partial  $\Sigma$ -algebra  $A$ , and define the notions of *computability*, *semicomputability* and *projective semicomputability* for these languages on  $A$ . Much of the material is taken from [14], adapted to partial algebras.

We begin by reviewing basic concepts: many-sorted signatures, algebras, and, in particular, topological partial algebras. Next we define the syntax and semantics of the **While** programming language. Then we extend this language with special programming constructs to form two new languages: **While**<sup>OR</sup> and **While**<sup>IN</sup>.

#### 3.1. Basic concepts: Signatures and algebras

A many-sorted signature  $\Sigma$  is a pair  $(\mathbf{Sort}(\Sigma), \mathbf{Func}(\Sigma))$  where

- (a)  $\mathbf{Sort}(\Sigma)$  is a finite set of ( $\Sigma$ -)sorts,  $s, s', \dots$   
 (b)  $\mathbf{Func}(\Sigma)$  is a finite set of basic ( $\Sigma$ -)function symbols

$$F : s_1 \times \dots \times s_m \rightarrow s \quad (m \geq 0).$$

The case  $m = 0$  gives a *constant symbol*; we then write  $F : \rightarrow s$ .

A ( $\Sigma$ -)product type has the form  $s_1 \times \dots \times s_m$  ( $m \geq 0$ ), where  $s_1, \dots, s_m$  are sorts. We write  $u, v, \dots$  for product types. A ( $\Sigma$ -)function type has the form  $u \rightarrow s$ , where  $u$  is a product type.

A  $\Sigma$ -algebra  $A$  has, for each  $\Sigma$ -sort  $s$ , a non-empty set  $A_s$ , the *carrier of sort  $s$* , and for each  $\Sigma$ -function symbol  $F : s_1 \times \dots \times s_m \rightarrow s$ , a (not necessarily total) function<sup>6</sup>

$$F^A : A^u \rightarrow A_s$$

where  $u = s_1 \times \dots \times s_m$ , and  $A^u = A_{s_1} \times \dots \times A_{s_m}$ .

We write  $\Sigma(A)$  for the signature of an algebra  $A$ .

**Example 3.1.1** (*Booleans*). The signature  $\Sigma(\mathcal{B})$  of booleans is

signature	$\Sigma(\mathcal{B})$
sorts	bool
functions	true, false : $\rightarrow$ bool, not : bool $\rightarrow$ bool or, and : bool <sup>2</sup> $\rightarrow$ bool, cor, cand : bool <sup>2</sup> $\rightarrow$ bool,

The algebra  $\mathcal{B}$  of booleans contains the carrier  $\mathbb{B} = \{\mathbf{t}, \mathbf{f}\}$  of sort bool, and the standard interpretations of the constant and function symbols of  $\Sigma(\mathcal{B})$ .

Note that  $\mathcal{B}$  contains two sets of boolean operators: (1) the strict operators 'or' and 'and'; and (2) the "conditional" operators 'cor' and 'cand' (denoted by '|' and '&&' in C-like languages), "evaluated from the left", and non-strict in the 2nd argument. These become important in the context of partial algebras such as  $\mathcal{R}$  (Example 3.1.5).

We will also use the infix notations ' $\vee$ ', ' $\wedge$ ' for the strict boolean operators or, and; and ' $\overset{c}{\vee}$ ', ' $\overset{c}{\wedge}$ ' for the "conditional" operators cor, cand.

<sup>6</sup> We use ' $\rightarrow$ ' to denote partial functions.

**Example 3.1.2** (*Naturals*). The signature of naturals is defined as

signature	$\Sigma(\mathcal{N})$
import	$\Sigma(\mathcal{B})$
sorts	nat
functions	0 : $\rightarrow$ nat, suc : nat $\rightarrow$ nat eq <sub>N</sub> , less <sub>N</sub> : nat <sup>2</sup> $\rightarrow$ bool

The corresponding algebra of naturals  $\mathcal{N}$  consists of the carrier  $\mathbb{N} = \{0, 1, 2, \dots\}$  of sort nat, the carrier  $\mathbb{B} = \{\mathbf{t}, \mathbf{f}\}$  of sort bool, and the standard constants and functions  $0_{\mathbb{N}} : \rightarrow \mathbb{N}$ ,  $\text{suc}_{\mathbb{N}} : \mathbb{N} \rightarrow \mathbb{N}$ , and  $\text{eq}_{\mathbb{N}}, \text{less}_{\mathbb{N}} : \mathbb{N}^2 \rightarrow \mathbb{B}$  (apart from the standard boolean operations).

We will use the infix notation ‘=’ and ‘<’ for ‘eq<sub>N</sub>’ and ‘less<sub>N</sub>’.

We come to the central concept of a topological partial algebra. First we note that for any two topological spaces  $X$  and  $Y$ , a partial function  $f : X \rightarrow Y$  is said to be *continuous* if for every open  $V \subseteq Y$ ,

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

is open in  $X$ . (This reduces to the usual notion of continuity when  $f$  is total.)

**Definition 3.1.3** (*Topological partial algebra*). A topological partial algebra is a partial  $\Sigma$ -algebra with topologies on the carriers such that each of the basic  $\Sigma$ -functions is continuous, and the carriers  $\mathbb{B}$  and  $\mathbb{N}$  (if present) have the discrete topology.

**Remark 3.1.4** (*Continuity of computable functions; the continuity principle*). The significance of the continuity of the basic functions of a topological algebra  $A$  is that it implies continuity of all **While** computable functions on  $A$ . This is the “Continuity Theorem” for topological algebras [13, §6], [14, §7.5].

This is in accordance with the *Continuity Principle* which can be expressed as

$$\text{computability} \implies \text{continuity}.$$

This principle is discussed in [12, Section 1] and [15, §3.1].<sup>7</sup>

**Example 3.1.5** (*Algebra of reals*). The signature of the algebra  $\mathcal{R}$  of reals is given by

signature	$\Sigma(\mathcal{R})$
import	$\Sigma(\mathcal{N})$
sorts	real
functions	0, 1 : $\rightarrow$ real, +, $\times$ : real <sup>2</sup> $\rightarrow$ real, − : real $\rightarrow$ real, eq <sub>R</sub> , less <sub>R</sub> : real <sup>2</sup> $\rightarrow$ bool

The corresponding algebra  $\mathcal{R}$  has the carrier  $\mathbb{R}$  of sort real, as well as the imported carriers  $\mathcal{N}$  and  $\mathcal{B}$ , of sort nat and bool, the real constants and operations (also written 0, 1, +,  $\times$ , −), and the boolean-valued partial functions  $\text{eq}_{\mathbb{R}} : \mathbb{R}^2 \rightarrow \mathbb{B}$  and  $\text{less}_{\mathbb{R}} : \mathbb{R}^2 \rightarrow \mathbb{B}$ , defined by:

$$\text{eq}_{\mathbb{R}}(x, y) = \begin{cases} \uparrow & \text{if } x = y \\ \mathbf{f} & \text{if } x \neq y \end{cases}$$

$$\text{less}_{\mathbb{R}}(x, y) = \begin{cases} \mathbf{t} & \text{if } x < y \\ \mathbf{f} & \text{if } x > y \\ \uparrow & \text{if } x = y. \end{cases}$$

Again we use the infix notation ‘=’ and ‘<’ for ‘eq<sub>R</sub>’ and ‘less<sub>R</sub>’.

<sup>7</sup> Cf. also the relationship between scientific observation and continuity, formulated as Hadamard’s Principle ([4,2], discussed also in [17]).

**Remarks 3.1.6** (Standard and  $N$ -standard algebras).

- (a) The algebras  $\mathcal{N}$  and  $\mathcal{R}$  are *standard*, in the sense that they contain the carrier  $\mathbb{B}$  with the standard boolean operations. Standardness of  $\mathcal{R}$  is necessary for the theoretical development in this paper. In fact we will assume that all algebras with which we deal are standard.
- (b)  $\mathcal{R}$  is also  *$N$ -standard*, in the sense that it contains the carrier  $\mathbb{N}$  with the standard arithmetic operations.  $N$ -standardness of  $\mathcal{R}$  is not really necessary for our main result, since the integers, and hence the naturals, can be implemented in the reals [14, Proposition 6.17]. However, it is a very useful assumption (see e.g. Section 3.9 below).

**Discussion 3.1.7** (Motivation for definition of partial functions). We want to motivate the definitions of partial functions in general, and more specifically, the functions  $\text{eq}_{\mathcal{R}}$  and  $\text{less}_{\mathcal{R}}$  in  $\mathcal{R}$ . We present our motivation in two ways: the first based on continuity considerations, and the second based on a “thought experiment” concerning (concrete) computation of the basic functions under discussion.

- (a) The total versions of  $\text{eq}_{\mathcal{R}}$  and  $\text{less}_{\mathcal{R}}$  are not continuous, as can easily be checked. (By contrast, the total functions  $\text{eq}_{\mathcal{N}}$ ,  $\text{less}_{\mathcal{N}}$  on  $\mathcal{N}$  are continuous, because of the discrete topology on  $\mathcal{N}$ .) Continuity of basic functions such as  $\text{eq}_{\mathcal{R}}$  and  $\text{less}_{\mathcal{R}}$ , making  $\mathcal{R}$  a topological algebra, is consistent with the Continuity Principle (see Remark 3.1.4).
- (b) Consider now a “thought experiment” involving the computation of an atomic formula  $x = y$ , where  $x$  and  $y$  are real variables. Suppose, at a particular state  $\sigma$ , we want to determine whether  $x = y$  is true. Suppose also (we are now combining “abstract” and “concrete” modes of description<sup>8</sup>) that the values of  $x$  and  $y$  at  $\sigma$  are “given by” Cauchy sequences of rationals  $(r_0, r_1, r_2, \dots)$  and  $(s_0, s_1, s_2, \dots)$ , which (for convenience) we assume to be “fast”, i.e.,

$$\forall n, \forall m \geq n \quad |r_n - r_m| < 2^{-n},$$

and similarly for  $(s_n)$ . Suppose also that for  $n = 0, 1, 2, \dots$  the inputs  $r_n$  and  $s_n$  are observed (from some device) at  $n$  time units. Now  $x < y$  is true at  $\sigma$  iff for some  $n$ ,  $r_n + 2 \cdot 2^{-n} < s_n$ , and this can be determined within a finite amount of time. Correspondingly,  $x = y$  is true iff for all  $n$ ,  $|r_n - s_n| \leq 2 \cdot 2^{-n}$ , but this cannot be determined within any finite amount of time, and so the evaluation of  $x = y$  diverges. These considerations explain the form of the partial definitions of equality and order on the reals.

### 3.2. Syntax of **Term**( $\Sigma$ )

**Definition 3.2.1** ( $\Sigma$ -variables). For each  $\Sigma$ -sort  $s$ ,  $\mathbf{Var}_s(\Sigma)$  is the set of  $\Sigma$ -variables  $x^s, y^s, \dots$  of sort  $s$ .

**Definition 3.2.2** ( $\Sigma$ -terms).  $\mathbf{Term}(\Sigma)$  is the set of  $\Sigma$ -terms  $t, \dots$ , and  $\mathbf{Term}_s(\Sigma)$  is the set of  $\Sigma$ -terms  $t^s, \dots$  of sort  $s$ , defined (in modified BNF) by

$$t^s ::= x^s \mid F(t_1^{s_1}, \dots, t_m^{s_m})$$

where  $F$  is a  $\Sigma$ -function symbol of type  $s_1 \times \dots \times s_m \rightarrow s$ .

We often drop the sort superscript  $s$ , and write  $t : s$  to indicate that  $t \in \mathbf{Term}_s(\Sigma)$ . More generally, we write  $t : u$  to indicate that  $t$  is a tuple of terms of product type  $u$ . We write  $\mathbf{Term}_s$  for  $\mathbf{Term}_s(\Sigma)$ , etc. We also write  $b, \dots$  for boolean  $\Sigma$ -terms, i.e.  $\Sigma$ -terms of sort  $\text{bool}$ .

### 3.3. Syntax of **While**( $\Sigma$ )

We will use ‘ $\equiv$ ’ to denote *syntactic identity* between two expressions.

**Definition 3.3.1** (Statements).  $\mathbf{Stmt}(\Sigma)$  is the class of *statements*  $S, \dots$  generated by:

$$S ::= \text{skip} \mid x := t \mid S_1 ; S_2 \mid \text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi} \mid \text{while } b \text{ do } S_0 \text{ od}$$

where the variable  $x$  and term  $t$  have the same  $\Sigma$ -sort.

**Definition 3.3.2** (Procedures).  $\mathbf{Proc}(\Sigma)$  is the class of *procedures*  $P, \dots$  of the form:

$$P \equiv \text{proc } D \text{ begin } S \text{ end}$$

<sup>8</sup> Recall the discussion in Section 1.1.

where the statement  $S$  is the *body* and  $D$  is a *variable declaration* of the form

$$D \equiv \text{in } a : u \text{ out } b : v \text{ aux } c : w$$

where  $a$ ,  $b$  and  $c$  are tuples of *input*, *output* and *auxiliary variables* respectively. We stipulate:

- (i)  $a$ ,  $b$ ,  $c$  each consist of distinct variables, and are pairwise disjoint,
- (ii) every variable occurring in  $S$  must be declared in  $D$ ,<sup>9</sup>
- (iii) all auxiliary and output variables are *initialized* with default values.

If  $a : u$  and  $b : v$ , then  $P$  is said to have type  $u \rightarrow v$ , written  $P : u \rightarrow v$ .

We turn to the semantics of terms, statements and procedures. Let  $A$  be a standard partial  $\Sigma$ -algebra.

### 3.4. States

**Definition 3.4.1** (*State*).

- (a) A *state* over  $A$  is a family  $\langle \sigma_s \mid s \in \mathbf{Sort}(\Sigma) \rangle$  of functions  $\sigma_s : \mathbf{Var}_s \rightarrow A_s$ .
- (b) **State**( $A$ ) is the set of states on  $A$ , with elements  $\sigma, \dots$

We write  $\sigma(x)$  for  $\sigma_s(x)$  where  $x : s$ . For a tuple  $x \equiv (x_1, \dots, x_m)$ , we write  $\sigma[x]$  for  $(\sigma(x_1), \dots, \sigma(x_m))$ .

**Definition 3.4.2** (*Variant of a state*). Let  $\sigma$  be a state over  $A$ , and for some  $\Sigma$ -product type  $u$ , let  $x \equiv (x_1, \dots, x_n) : u$  and  $a = (a_1, \dots, a_n) \in A^u$  (for  $n \geq 1$ ). We define  $\sigma\{x/a\}$  to be the state over  $A$  formed from  $\sigma$  by replacing its value at  $x_i$  by  $a_i$  for  $i = 1, \dots, n$ .

### 3.5. Semantics of terms

For  $t \in \mathbf{Term}_s$ , we will define the function

$$\llbracket t \rrbracket^A : \mathbf{State}(A) \rightarrow A_s$$

where  $\llbracket t \rrbracket^A \sigma$  is the value of  $t$  in  $A$  at state  $\sigma$ .

**Notation 3.5.1.**

- (a)  $\llbracket t \rrbracket^A \sigma \downarrow$  means that evaluation of  $\llbracket t \rrbracket^A \sigma$  halts, or converges; and  $\llbracket t \rrbracket^A \sigma \downarrow a$  means that it converges to a value  $a$ .
- (b)  $\llbracket t \rrbracket^A \sigma \uparrow$  means that evaluation of  $\llbracket t \rrbracket^A \sigma$  diverges.

**Notation 3.5.2** (*Kleene equality*). We write e.g.

$$\llbracket t_1 \rrbracket^A \sigma \simeq \llbracket t_2 \rrbracket^A \sigma$$

to mean that the two sides of the equality either both converge to the same value, or both diverge [6, §63]

**Definition 3.5.3** (*Semantics of terms*). The definition of  $\llbracket t \rrbracket^A \sigma$  is by structural induction on  $\Sigma$ -terms  $t$ :

$$\begin{aligned} \llbracket x \rrbracket^A \sigma &= \sigma(x) \\ \llbracket F(t_1, \dots, t_m) \rrbracket^A \sigma &\simeq \begin{cases} F^A(\llbracket t_1 \rrbracket^A \sigma, \dots, \llbracket t_m \rrbracket^A \sigma) & \text{if } \llbracket t_i \rrbracket^A \sigma \downarrow \text{ for } 1 \leq i \leq m \\ \uparrow & \text{otherwise.} \end{cases} \end{aligned}$$

Note that if  $c : \rightarrow s$ , i.e.,  $c$  is a constant symbol of sort  $s$ , then  $\llbracket c \rrbracket^A \sigma = c^A \in A_s$ .

**Definition 3.5.4** (*Semantic equivalence of terms*). Two  $\Sigma$ -terms  $t_1$  and  $t_2$  of the same sort  $s$  are (*semantically*) *equivalent over*  $A$ , written  $t_1 \approx t_2$ , iff

$$\forall \sigma \in \mathbf{State}(A) \quad (\llbracket t_1 \rrbracket^A \sigma \simeq \llbracket t_2 \rrbracket^A \sigma).$$

<sup>9</sup> This will not hold for the auxiliary variable in the ‘Exist’ construct (Section 3.9).



**Definition 3.5.5** (Weak semantic equivalence of booleans). Two  $\Sigma$ -booleans  $b_1$  and  $b_2$  are weakly (semantically) equivalent over  $A$ , written  $b_1 \sim b_2$ , iff

$$\forall \sigma \in \mathbf{State}(A) \quad (\llbracket b_1 \rrbracket^A \sigma \downarrow \mathbf{t} \iff \llbracket b_2 \rrbracket^A \sigma \downarrow \mathbf{t}).$$

### 3.6. Semantics of statements

The meaning  $\llbracket S \rrbracket^A$  of a **While**( $\Sigma$ ) statement  $S$  is a partial state transformer on an algebra  $A$ :

$$\llbracket S \rrbracket^A : \mathbf{State}(A) \rightharpoonup \mathbf{State}(A).$$

Its definition is standard [13,14] and lengthy, and so we omit it.

Briefly, it is based on defining the *computation sequence* of  $S$  starting in a state  $\sigma$ , or rather the  $n$ -th component of this sequence, by a primary induction on  $n$ , and a secondary induction on the size of  $S$ .

The following results show that the i/o semantics for statements  $S$  satisfies certain desirable properties, which will be used later.

#### Lemma 3.6.1.

(i) For  $S$  atomic:  $S \equiv \text{skip}$  or  $S \equiv x := t$ ,

$$\begin{aligned} \llbracket \text{skip} \rrbracket^A \sigma &= \sigma \\ \llbracket x := t \rrbracket^A \sigma &\simeq \sigma \{x / \llbracket t \rrbracket^A \sigma\}. \end{aligned}$$

(ii) If  $S \equiv S_1; S_2$ ,

$$\llbracket S \rrbracket^A \sigma \simeq \llbracket S_2 \rrbracket^A (\llbracket S_1 \rrbracket^A \sigma).$$

(iii) If  $S \equiv \text{if } b \text{ then } S_1 \text{ else } S_2 \text{ fi}$ ,

$$\llbracket S \rrbracket^A \sigma \simeq \begin{cases} \llbracket S_1 \rrbracket^A \sigma & \text{if } \llbracket b \rrbracket^A \sigma \downarrow \mathbf{t} \\ \llbracket S_2 \rrbracket^A \sigma & \text{if } \llbracket b \rrbracket^A \sigma \downarrow \mathbf{f} \\ \uparrow & \text{if } \llbracket b \rrbracket^A \sigma \uparrow. \end{cases}$$

(iv) If  $S \equiv \text{while } b \text{ do } S_0 \text{ do}$ ,

$$\llbracket S \rrbracket^A \sigma \simeq \begin{cases} \llbracket S_0; S \rrbracket^A \sigma & \text{if } \llbracket b \rrbracket^A \sigma \downarrow \mathbf{t} \\ \sigma & \text{if } \llbracket b \rrbracket^A \sigma \downarrow \mathbf{f} \\ \uparrow & \text{if } \llbracket b \rrbracket^A \sigma \uparrow. \end{cases}$$

**Proof.** As outlined in [14, Theorem 3.6] adapted to partial algebras.  $\square$

### 3.7. Semantics of procedures

If

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

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

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

defined as follows. For  $a \in A^u$ , let  $\sigma$  be any state on  $A$  such that  $\sigma[a] = a$ , and  $\sigma[b]$  and  $\sigma[c]$  are given suitable default values. Then

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

Note that  $P^A$  is well defined, by the functionality lemma for statement semantics [14, Lemma 3.10].

### 3.8. **While** computability

**Definition 3.8.1** (*While* computable function).

- (a) A function  $f$  on  $A$  of type  $u \rightarrow v$  is said to be *computable on  $A$  by a **While** procedure*  $P : u \rightarrow v$  if  $f = P^A$ .  
 (b)  $\mathbf{While}(A)$  is the class of functions **While** computable on  $A$ .

**Definition 3.8.2** (*Halting set*). The *halting set* of a procedure  $P : u \rightarrow v$  on  $A$  is the set

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

**Definition 3.8.3** (*While* semicomputable set). A set  $R \subseteq A^u$  is **While** semicomputable on  $A$  if it is the halting set on  $A$  of some **While** procedure.

**Definition 3.8.4** (*Projectively While* semicomputable set). A set  $R \subseteq A^u$  is *projectively While* semicomputable on  $A$  iff  $R$  is the projection of a **While** semicomputable set on  $A$ , i.e., for some product types  $u$  and  $v$ , there is a **While** semicomputable set  $R' \subseteq A^{u \times v}$  such that

$$\forall x \in A^u \quad [x \in R \iff \exists y \in A^v : (x, y) \in R'].$$

Generally, projective semicomputability is a more powerful (and less algorithmic) concept than semicomputability. (But see [Theorem 5](#) in Section 5.)

### 3.9. Expanding **While** to $\mathbf{While}^{\text{OR}}$ and $\mathbf{While}^{\text{IN}}$

Let  $\Sigma$  be a standard signature. Recall (from [Example 3.1.1](#)) that it contains both the strict boolean operators  $\vee$ ,  $\wedge$ , and the “conditional” operators  $\overset{c}{\vee}$ ,  $\overset{c}{\wedge}$ .

Now we consider the addition to  $\Sigma$  of a third pair of boolean operators: the strong “Kleene operators”

$$\text{OR, AND} : \text{bool}^2 \rightarrow \text{bool}$$

[6, p. 334], which are non-strict in both arguments. We will use the infix notation ‘ $\nabla$ ’ and ‘ $\Delta$ ’ for these.

The ‘OR’ operator allows us to simulate *interleaving* at an abstract level, since it lets us decide a disjunction  $b_1 \nabla b_2$  of two boolean terms to be true if either of these converges to  $\mathbf{t}$  (even if the other one diverges).

Let  $\Sigma^{\text{OR}}$  be the expansion of  $\Sigma$  formed by adding ‘OR’. We then define:

$$\mathbf{Term}^{\text{OR}}(\Sigma) = \mathbf{Term}(\Sigma^{\text{OR}})$$

$$\mathbf{Bool}^{\text{OR}}(\Sigma) = \mathbf{Bool}(\Sigma^{\text{OR}})$$

$$\mathbf{While}^{\text{OR}}(\Sigma) = \mathbf{While}(\Sigma^{\text{OR}}).$$

We can also extend the **While** language by adding a new boolean term

$$\exists z P(t, z)$$

where the procedure  $P$  has type  $u \times \text{nat} \rightarrow \text{bool}$ , and  $z$  is a “new” variable of sort  $\text{nat}$ . This will occur only in the context:

$$x^B := \exists z P(t, z).$$

We define its semantics as:

$$\llbracket \exists z P(t, z) \rrbracket^A \sigma \simeq \begin{cases} \mathbf{t} & \text{if } P^A(\llbracket t \rrbracket^A \sigma, n) \downarrow \mathbf{t} \text{ for some } n \\ \uparrow & \text{otherwise.} \end{cases} \quad (3.1)$$

This corresponds to the following operational semantics: interleave the computations for

$$P^A(t, 0), P^A(t, 1), P^A(t, 2), \dots$$

and return  $\mathbf{t}$  if and only if any of these procedures terminates and returns  $\mathbf{t}$ ; otherwise keep on going.

This operation allows us to simulate *infinite interleaving* at the abstract level.

Note that this is different from “evaluating from the left”, which can be implemented by a simple loop:

```

find := false;
z := 0;
while find = false
do
  find := P(t, z)
  z := z + 1;
od

```

which will *diverge* in case, e.g.,

$$P^A(t, 0) \downarrow \mathbf{f}, \quad P^A(t, 1) \uparrow, \quad P^A(t, 2) \downarrow \mathbf{t}$$

whereas  $\exists z P(t, z)$  will converge to  $\mathbf{t}$ .

The usefulness of these new program constructs will become apparent in Section 4.

Using the ‘Exist’ construct, we can “weakly simulate” OR, i.e., define a procedure  $P$  such that

$$\text{Exist } z : P(b_1, b_2, z) \sim b_1 \nabla b_2$$

(recall Definition 3.5.5). In fact, we can define  $P(b_1, b_2, z)$  as

```

proc
in  b1, b2 : bool
   z : nat
out b : bool
begin
  b := if z = 1 then b1 else
        if z = 2 then b2 else
        false
      fi
    fi
end.

```

Note that  $\text{Exist } z : P(b_1, b_2, z)$  is only weakly semantically equivalent to  $b_1 \nabla b_2$ ; in fact no construct of the form  $\text{Exist } z : P(b_1, b_2, z)$  can be strongly equivalent to  $b_1 \nabla b_2$ , since when  $b_1$  and  $b_2$  both have the value  $\mathbf{f}$ , then  $b_1 \nabla b_2$  has the value  $\mathbf{f}$ , but  $\text{Exist } z : P(b_1, b_2, z)$  can only have values  $\mathbf{t}$  and  $\uparrow$ , by (3.1).

We can nevertheless think of ‘OR’ as a “finite” version of ‘Exist’, and so we adjoin the ‘OR’ construct together with ‘Exist’ to form the language  $\mathbf{While}^{\exists\mathbf{N}}(\Sigma)$ .

We write  $\Sigma^{(\text{OR})}$  for the signature  $\Sigma$  or  $\Sigma^{\text{OR}}$ , and similarly  $\mathbf{While}^{(\text{OR})}$  for the language  $\mathbf{While}$  or  $\mathbf{While}^{\text{OR}}$ .

**Remark 3.9.1** (Continuity of  $\mathbf{While}^{\text{OR}}$  and  $\mathbf{While}^{\exists\mathbf{N}}$  computable functions). As stated above (Remark 3.1.4) all  $\mathbf{While}$  computable functions on a topological partial algebra are continuous. The same applies to  $\mathbf{While}^{\text{OR}}$  and  $\mathbf{While}^{\exists\mathbf{N}}$  computable functions. We omit proofs. Again, this is important because of the Continuity Principle.

**Remark 3.9.2.** The ‘Exist’ construct can be implemented from the ‘choose’ construct (or “countable choice” operator) [15] by

$$x^{\text{B}} := \exists z P(t, z) \iff n := \text{choose } z : P(t, z); x^{\text{B}} := P(t, n).$$

However, unlike the ‘choose’ construct which is nondeterministic, the ‘Exist’ construct is “weakly” or “globally” deterministic, i.e., deterministic at the abstract level, although there is nondeterminism in the actual choice of  $z$  in a concrete implementation.

Clearly,  $\mathbf{While}$  computability implies  $\mathbf{While}^{\text{OR}}$  computability, which in turn implies  $\mathbf{While}^{\exists\mathbf{N}}$  computability.

### 3.10. $\mathbf{While}_0$ language

To simplify the formal development in the next section, we restrict the structure of  $\mathbf{While}$  statements to a special form, and show that all statements can be effectively transformed to this form.

**Definition 3.10.1** (Special form for  $\mathbf{While}$  statements). A  $\mathbf{While}(\Sigma)$  statement  $S$  is said to be in *special form* if (inductively) it has one of the following forms:

- $S \equiv \text{skip}$
- $S \equiv x^s := t^s$
- $S \equiv \text{if } x^B \text{ then } S_1 \text{ else } S_2 \text{ fi}$
- $S \equiv \text{while } x^B \text{ do } S_0 \text{ od}$
- $S \equiv S_1; S_2$

where  $S_0, S_1$  and  $S_2$  are also in *special form*.

In other words,  $S$  is in special form iff all *boolean tests* occurring in  $S$  are *variables*.

Let  $\mathbf{While}_0(\Sigma)$  be the  $\mathbf{While}(\Sigma)$  language restricted to special form; and similarly for  $\mathbf{While}_0^{\text{OR}}(\Sigma)$  and  $\mathbf{While}_0^{\text{IN}}(\Sigma)$ .

### Lemma 3.10.2.

- (a) All **While** statements can be effectively transformed into  $\mathbf{While}_0$  statements, preserving the semantics.  
 (b) Similarly for  $\mathbf{While}^{\text{OR}}$  and  $\mathbf{While}^{\text{IN}}$ .

The proofs are quite routine.

### Remarks 3.10.3.

- (a) In the  $\mathbf{While}_0^{\text{IN}}$  language, there are two kinds of assignment: the ‘Exist’ assignment, of the form

$$x^B := \exists z P(t, z)$$

and all other assignments

$$x := t$$

which we call *simple* assignments.

- (b) In  $\mathbf{While}_0, \mathbf{While}_0^{\text{OR}}$  or  $\mathbf{While}_0^{\text{IN}}$  statements, the only way for a program to diverge locally is by the divergence of the right-hand side of an assignment statement.  
 (c) From now on, we will only work with  $\mathbf{While}_0, \mathbf{While}_0^{\text{OR}}$  and  $\mathbf{While}_0^{\text{IN}}$  programs. To simplify the notation, we will still refer to these as **While** (etc.) programs.

#### 3.11. Definability property

This is needed in the construction of the computation tree and in the proof of Engeler’s Lemma in the next section.

#### Definition 3.11.1 (Definability predicate).

- (a) A *definability predicate* at sort  $s$  for a  $\Sigma$ -algebra  $A$  is a  $\Sigma$ -boolean expression  $\mathbf{def}_s$ , containing a distinguished free variable  $x : s$ , such that for all  $\Sigma$ -terms  $t$  and all states  $\sigma$  on  $A$  (writing  $\mathbf{def}_s(t)$  for  $\mathbf{def}_s(x/t)$ ):

$$\llbracket \mathbf{def}_s(t) \rrbracket^A \sigma \simeq \begin{cases} \mathbf{tt} & \text{if } \llbracket t \rrbracket^A \sigma \downarrow \\ \uparrow & \text{otherwise.} \end{cases}$$

- (b) A  $\Sigma$ -algebra has the *definability property* if it has a definability predicate at all  $\Sigma$ -sorts.

For all the algebras  $A$  with which we deal, we assume:

#### Assumption 3.11.2 (Definability). $A$ has the definability property.

In particular, we show:

#### Lemma 3.11.3. $\mathcal{R}$ has the definability property.

**Proof.** In  $\Sigma(\mathcal{R})$ , we can define  $\mathbf{def}_s(t)$  as follows:

At sort  $\text{nat}$ , put  $\mathbf{def}_{\text{nat}}(t) \equiv \text{true}$ .

At sort  $\text{real}$ , put  $\mathbf{def}_{\text{real}}(t) \equiv \text{less}_{\mathbb{R}}(t, t + 1)$ .

For the boolean term  $\exists z P(t, z)$ , put

$$\mathbf{def}_{\text{bool}}(\exists z P(t, z)) \equiv \exists z P(t, z).$$

For any other term  $t$  of sort  $\text{bool}$ , put  $\mathbf{def}_{\text{bool}}(t) \equiv (t \vee \neg t)$ .  $\square$

### 4. Computation trees; Engeler’s Lemma

Engeler’s Lemma [3] is an important theoretical tool for the research described in this paper. It states (roughly) that a semicomputable set can be expressed as the disjunction of an effective infinite sequence of booleans.

A proof of Engeler’s Lemma for the **While** language on total algebras was given in [14, §5]. Here we prove Engeler’s Lemma for the **While**, **While**<sup>OR</sup> and **While**<sup>3N</sup> languages on partial algebras. Our proof is based on computation trees (in the case of **While**<sup>(OR)</sup>) and “hypertrees” (in the case of **While**<sup>3N</sup>).

We also prove a Semantic Disjointedness Lemma 4.4.2 which will play a central role in our Structure Theorems.

#### 4.1. Computation tree for **While**<sup>(OR)</sup>(Σ)

We define a computation tree  $\mathcal{T}[S, \mathbf{x}]$  for a **While**<sup>(OR)</sup> statement  $S$  on  $\mathcal{R}$ , where  $\mathbf{Var}(S) \subseteq \mathbf{x} \equiv (x_1, \dots, x_n) : u$ . The computation tree  $\mathcal{T}[S, \mathbf{x}]$  is like an “unfolded flow chart” for  $S$ .

This is a version of the computation tree defined in [14, §5.10], adapted for the **While**<sup>(OR)</sup> languages and for partial algebras.

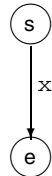
The root of  $\mathcal{T}[S, \mathbf{x}]$  is labeled ‘s’ (for ‘start’), and the leaves are labeled ‘e’ (for ‘end’). The internal nodes are labeled with assignments and boolean tests. Each edge is labeled with a syntactic state, i.e., a tuple of terms  $t \equiv (t_1, \dots, t_n) : u$ . The idea is that if  $S$  is executed at an initial state  $\sigma$ , then the state at this point of the computation will be  $\sigma\{\mathbf{x}/\llbracket t \rrbracket^A \sigma\}$ .

In the course of the following definition we will make use of the restricted tree  $\mathcal{T}^-[S, \mathbf{x}]$ , which is just  $\mathcal{T}[S, \mathbf{x}]$  without the ‘s’ node.

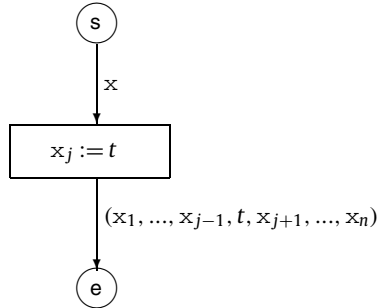
We will also use the notation  $\mathcal{T}[S, t]$  for the tree formed from  $\mathcal{T}[S, \mathbf{x}]$  by replacing all edges labeled  $t'$  (say) by  $t' \langle \mathbf{x}/t \rangle$ .

The definition of  $\mathcal{T}[S, \mathbf{x}]$  is by structural induction on  $S$ .

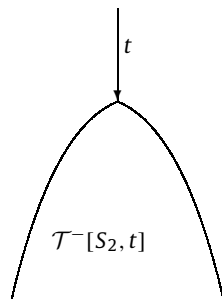
(1)  $S \equiv \text{skip}$ . Then  $\mathcal{T}[S, \mathbf{x}]$  is just



(2)  $S \equiv x_j := t$ . Then  $\mathcal{T}[S, \mathbf{x}]$  is the tree



(3)  $S \equiv S_1; S_2$ . Then  $\mathcal{T}[S, \mathbf{x}]$  is formed from  $\mathcal{T}[S_1, \mathbf{x}]$  by replacing each leaf in a state  $t$  by the tree



(4)  $S \equiv \text{if } x^B \text{ then } S_1 \text{ else } S_2 \text{ fi}$ . Then  $\mathcal{T}[S, \mathbf{x}]$  is shown in Fig. 1.

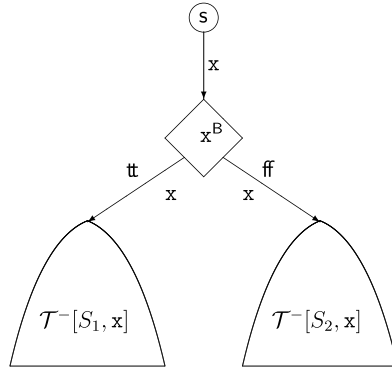


Fig. 1. Construction of  $\mathcal{T}[S, x]$  (case 4).

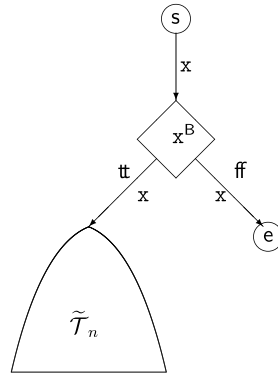


Fig. 2. Construction of  $\mathcal{T}[S, x]$  (case 5).

(5)  $S \equiv \text{while } x^B \text{ do } S_0 \text{ od}$ . Then  $\mathcal{T}[S, x]$  is defined as the “limit” of the sequence of trees  $\mathcal{T}_n[S, x]$ , where  $\mathcal{T}_0[S, x]$  is  $\mathcal{T}[\text{skip}, x]$ , as in (1) above, and  $\mathcal{T}_{n+1}[S, x]$  is as shown in Fig. 2, where  $\tilde{\mathcal{T}}_n$  is the tree formed from  $\mathcal{T}^-[S_0, x]$  by replacing each leaf in a state  $t$  by  $\mathcal{T}_n^-[S, t]$ .

Note that the construction of  $\mathcal{T}[S, x]$  is *effective* in  $S$  and  $x$ . More precisely:  $\mathcal{T}[S, x]$  can be coded as an r.e. set of numbers, with r.e. index primitive recursive in  $\ulcorner S \urcorner$  and  $\ulcorner x \urcorner$ .

#### 4.2. Semantics of infinite disjunctions

We will show that the halting set of a **While**, **While**<sup>OR</sup> or **While**<sup>BN</sup> procedure can be expressed as the countable disjunction of an effective infinite sequence of booleans. We must therefore first consider carefully some possible semantics for infinite disjunctions in 3-valued logic.

Let  $b_k$  be a sequence of  $\Sigma^{\text{OR}}$ -booleans. There are (at least) two different reasonable semantics for the infinite disjunction

$$\bigvee_{k=0}^{\infty} b_k$$

for 3-valued logics (“reasonable” in the sense of having computational significance), for which we use distinct notations:

(1) *Infinite conditional disjunction* (“evaluation from the left”), denoted  $\bigvee_{k=0}^{\infty} b_k$ , with two possible outputs,  $\mathbf{t}$  and  $\uparrow$ :

$$\left[ \bigvee_{k=0}^{\infty} b_k \right]^A \sigma \simeq \begin{cases} \mathbf{t} & \text{if } \exists k: \llbracket b_k \rrbracket^A \sigma \downarrow \mathbf{t} \wedge \forall i < k \llbracket b_i \rrbracket^A \sigma \downarrow \mathbf{f} \\ \uparrow & \text{otherwise.} \end{cases}$$

This definition is **While** computable (in the sequence  $\ulcorner b_k \urcorner$ ) with the following procedure:

Evaluate  $b_k$  ( $k = 0, 1, \dots$ ) one by one. There are 3 possibilities:

- for some  $k$ , evaluation of  $b_k$  converges to  $\mathbf{t}$ , and all earlier  $b_j$  converge to  $\mathbf{f}$ , or
- for some  $k$ , evaluation of  $b_k$  diverges, and all earlier  $b_j$  converge to  $\mathbf{f}$  (“local divergence”), or
- all the  $b_k$  converge to  $\mathbf{f}$  (“global divergence”).

In the first case, evaluation of the disjunction converges to  $\mathbf{t}$ .

In the latter two cases, it diverges.

(2) *Infinite strong disjunction* (“strong Kleene evaluation”), denoted  $\bigvee_{k=0}^{\infty} b_k$ , again with two possible outputs,  $\mathbf{t}$  and  $\uparrow$ :

$$\left[ \bigvee_{k=0}^{\infty} b_k \right]^A \sigma \simeq \begin{cases} \mathbf{t} & \text{if } \exists k \llbracket b_k \rrbracket^A \sigma \downarrow \mathbf{t} \\ \uparrow & \text{otherwise.} \end{cases}$$

This definition is not (in general) **While**<sup>(OR)</sup> computable (in  $\ulcorner b_k \urcorner$ ), but it is **While**<sup>3N</sup> computable, by the semantics of  $\exists z P(t, z)$  (Section 3.9).

Definition (2) is the one mainly used in this paper, e.g. in the formulation of Engeler’s Lemma (Lemma 4.3.1 below). Intuitively, definitions (1) and (2) generalize (respectively) the finite disjunctions ‘ $\overset{c}{\vee}$ ’ and ‘ $\nabla$ ’.

**Notation 4.2.1.** For any boolean term  $b$  with  $\mathbf{Var}(b) \subseteq x : u$ , and  $a \in A^u$ , we write  $b[a]$  to mean:  $\llbracket b \rrbracket^A \sigma \downarrow \mathbf{t}$  for any  $\sigma \in \mathbf{State}(A)$  such that  $\sigma[x] = a$ .

**Definition 4.2.2** (Relation defined by boolean). A  $\Sigma^{(\text{OR})}$ -boolean term  $b$  with  $\mathbf{Var}(b) \subseteq x : u$  is said to define a relation  $R \subseteq A^u$  (w.r.t.  $x$ ) iff for all  $a \in A^u$

$$a \in R \iff b[a].$$

#### 4.3. Engeler’s Lemma for **While**<sup>(OR)</sup>

**Lemma 4.3.1** (Engeler’s Lemma for **While**<sup>(OR)</sup>). If a relation  $R \subseteq A^u$  is **While**<sup>(OR)</sup> semicomputable over a standard partial  $\Sigma$ -algebra  $A$ , then  $R$  can be expressed as the (strong) disjunction of an effective sequence of  $\Sigma^{(\text{OR})}$ -booleans over  $A$ .

**Proof.** Suppose  $R$  is the halting set in  $A$  of the **While**<sup>(OR)</sup> procedure:

$$P \equiv \text{proc in } a \text{ out } b \text{ aux } c \text{ begin } S \text{ end.} \quad (4.1)$$

For each leaf  $\lambda$  of the computation tree  $\mathcal{T}[S, x]$  there is a boolean  $b_{S,\lambda}$  with variables among  $x \equiv (a, b, c)$  which expresses the conjunction (“and”) of the test results and definability predicates along the path from the root to  $\lambda$ , as follows.

There are two cases to consider, according to the kind of node encountered along the path: assignment nodes and test nodes.

(1) An *assignment node*  $x^S := t^S$  in the path contributes to  $b_{S,\lambda}$  the conjunct expressing definability of the term  $t$ :

$$\dots \overset{c}{\wedge} \mathbf{def}_S(t) \overset{c}{\wedge} \dots$$

which guarantees that the boolean term  $b_{S,\lambda}$  converges only if evaluation of  $t$  converges at that point.

(2) A *test node* labeled  $x^B$  contributes as conjunct

$$\text{either } \dots \overset{c}{\wedge} x^B \overset{c}{\wedge} \dots$$

$$\text{or } \dots \overset{c}{\wedge} \neg x^B \overset{c}{\wedge} \dots$$

according to whether the path goes to the left or right here. Note that since the boolean test only has the form of a boolean variable  $x^B$ , we do not need to add the  $\mathbf{def}_{\text{bool}}$  predicate here.

Next, we can *effectively enumerate* the leaves of the computation tree to obtain a sequence  $(\lambda_k)$  by (for example) increasing the depth, and, at a given depth, going from left to right. (To ensure that the corresponding sequence of booleans  $(b_{S,\lambda_k})$  is infinite, we can “pad” it with the default value false.) Then for all  $a \in A^u$  (putting  $b_{S,k} \equiv b_{S,\lambda_k}$ ):

$$a \in R \iff P^A(a) \downarrow \iff \bigvee_{k=0}^{\infty} b_{S,k}[a].$$

Note we are using “infinite strong disjunction” (version (2) in Section 4.2).

Hence  $R$  can be expressed as the infinite strong disjunction of an effective countable sequence of  $\Sigma^{(\text{OR})}$ -booleans over  $A$ .  $\square$

#### 4.4. Semantic disjointness

**Definition 4.4.1** (*Semantic disjointness*). A sequence  $(b_0, b_1, b_2, \dots)$  of boolean terms is *semantically disjoint* over  $A$  if for any state  $\sigma$  on  $A$  and any  $n$ ,

$$\llbracket b_n \rrbracket^A \sigma \downarrow \mathbf{t} \implies \forall i \neq n, \llbracket b_i \rrbracket^A \sigma \downarrow \mathbf{f}.$$

**Lemma 4.4.2** (*Semantic Disjointness Lemma*). The sequence of boolean terms generated from a **While**<sup>(OR)</sup> computation tree  $S$  as in the proof of Engeler's [Lemma 4.3.1](#) is semantically disjoint.

**Proof.** Let  $i, j$  be distinct natural numbers and

$$b_{S,i} \equiv b_{S,i_1} \overset{c}{\wedge} \dots \overset{c}{\wedge} b_{S,i_m} \tag{4.2a}$$

$$b_{S,j} \equiv b_{S,j_1} \overset{c}{\wedge} \dots \overset{c}{\wedge} b_{S,j_n}. \tag{4.2b}$$

Note that for any  $k$ , the definition of  $b_{S,k}$  determines a path from the root to the  $k$ -th leaf of the computation tree of  $S$ . Therefore, considering the paths from the root to the  $i$ -th leaf and from the root to the  $j$ -th leaf, there must be a branching node with label  $b$  (say), where the two paths split, i.e. there exists some  $l < \min(m, n)$  such that

$$b_{S,i_1} \equiv b_{S,j_1}, \quad b_{S,i_2} \equiv b_{S,j_2}, \quad \dots, \quad b_{S,i_{(l-1)}} \equiv b_{S,j_{(l-1)}} \tag{4.3}$$

and

$$\begin{aligned} &\text{either } (b_{S,i_l} \equiv b \text{ and } b_{S,j_l} \equiv \neg b) \\ &\text{or } (b_{S,i_l} \equiv \neg b \text{ and } b_{S,j_l} \equiv b). \end{aligned} \tag{4.4}$$

So for any  $\sigma$ , suppose

$$\llbracket b_{S,i} \rrbracket^A \sigma = \mathbf{t}.$$

Then from (4.2a),

$$\llbracket b_{S,i_l} \rrbracket^A \sigma = \mathbf{t}$$

and from (4.4),

$$\llbracket b_{S,j_l} \rrbracket^A \sigma = \mathbf{f}. \tag{4.5}$$

Also, since by (4.3)

$$\llbracket b_{S,j_k} \rrbracket^A \sigma = \llbracket b_{S,i_k} \rrbracket^A \sigma = \mathbf{t} \quad \text{for all } k < l \tag{4.6}$$

then by (4.2b), (4.4) and (4.6) and the semantics of  $\overset{c}{\wedge}$ ,

$$\llbracket b_{S,j} \rrbracket^A \sigma = \mathbf{t} \overset{c}{\wedge} \dots \overset{c}{\wedge} \mathbf{t} \overset{c}{\wedge} \mathbf{f} \overset{c}{\wedge} \dots = \mathbf{f}. \quad \square$$

**Lemma 4.4.3** (*Semantic disjointness evaluation*). If an effective sequence of booleans  $(b_k)$  is semantically disjoint over  $A$ , then<sup>10</sup>

$$\bigvee_{k=0}^{\infty} b_k \approx \bigvee_{k=0}^{\overset{c}{\infty}} b_k \tag{4.7}$$

i.e., for any  $\sigma$ ,  $\llbracket \bigvee_k b_k \rrbracket^A \sigma$  can be “evaluated from the left”.

**Proof.** For any  $\sigma$ , we consider two cases:

- (1) There exists  $k$  such that  $\llbracket b_k \rrbracket^A \sigma \downarrow \mathbf{t}$ . Then by [Definition 4.4.1](#) of semantic disjointness,  $\llbracket b_i \rrbracket^A \sigma \downarrow \mathbf{f}$  for all  $i \neq k$ , and in particular for all  $i < k$ . Hence both sides of (4.7) converge to  $\mathbf{t}$  at  $\sigma$ .
- (2) Otherwise: as is easily seen, both sides diverge at  $\sigma$ .  $\square$

<sup>10</sup> Recall the notation ‘ $\approx$ ’ for semantic equivalence ([Definition 3.5.4](#)).



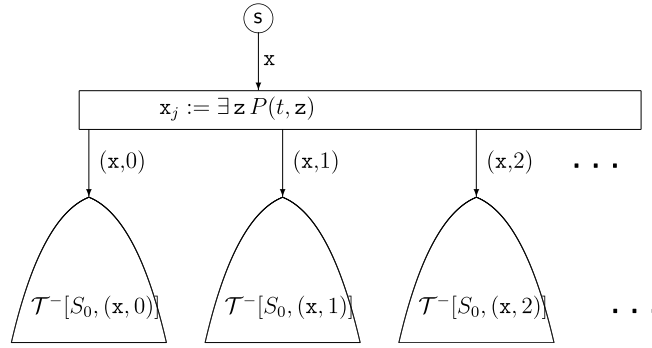


Fig. 3. Construction of hypertree (step 1).

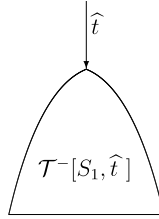


Fig. 4. Construction of hypertree (step 2).

4.5. Computation tree for **While**<sup>3N</sup>

In order to prove Engeler’s Lemma for the **While**<sup>3N</sup> language, we define (inductively) the computation trees for **While**<sup>3N</sup> statements, following the cases in the definition of the **While**<sup>(OR)</sup> computation tree in Section 4.1. We add a new case to the cases considered there:

(6)  $S \equiv x_j := \exists z P(t, z); S_1$ , where

$P \equiv \text{proc in } a \text{ out } b \text{ aux } c \text{ begin } S_0 \text{ end.}$

If  $S \equiv x_j := \exists z P(t, z)$  (with  $S_1$  missing), we just let  $S_1 \equiv \text{skip}$ .

The tree for  $S$  is then formed from the tree in Fig. 3 by replacing each ‘e’ leaf of the trees  $T^-[S_0, (x, \bar{i})]$  ( $i = 0, 1, 2, \dots$ ) by the tree in Fig. 4, where

$$\hat{t} =_{df} (x_1, \dots, x_{j-1}, \text{true}, x_{j+1}, \dots, x_n),$$

and then collapsing these multiple occurrences of the subtree  $T^-[S_1, \hat{t}]$  to form the tree shown in Fig. 5.

We call the subtrees  $T^-[S_0, (x, \bar{i})]$  ( $i = 0, 1, 2, \dots$ ) appearing in Fig. 5 *proc-subtrees* of the whole computation tree.

Define a *channel* in the tree of Fig. 5 to be a path through one of the “former leaves” of the proc-subtree  $T^-[S_0, (x, \bar{i})]$ , labeled  $c_{i,j}$ , where ‘ $i$ ’ refers to the  $i$ -th proc-subtree, and ‘ $j$ ’ refers to the  $j$ th “former leaf” of the proc-subtree. Note that in this tree, there are (countably) infinitely many channels from the root to the subtree  $T^-[S_1, \hat{t}]$ . We can effectively enumerate these channels by renaming channel  $c_{i,j}$  as  $c_k$  where  $k = \ulcorner(i, j)\urcorner$ .

Let  $\mathcal{T}[S, x]$  be the computation tree defined as above. Strictly speaking,  $\mathcal{T}[S, x]$  is not a tree, but a dag (directed acyclic graph). Call the node for  $x_j := \exists z P(t, z)$  shown in Fig. 5, together with the subgraph below it (excluding the subtree  $T^-[S_1, \hat{t}]$ ), the *hypernode* for  $x_j := \exists z P(t, z)$ ; and call the whole tree, constructed in this way, a *hypertree*. We can reduce such hypernodes to “atomic nodes” by ignoring their internal details, and so reduce the hypertree to a *reduced tree* (that is an actual tree, not a dag), just like the **While**<sup>(OR)</sup> computation tree constructed in Section 4.1.

Notice that there are no leaves in the proc-subtrees because we have replaced all the leaves by the subtree  $T^-[S_1, \hat{t}]$ . So the leaves of the hypertree can be identified with the leaves of the corresponding reduced tree, and hence they can be effectively enumerated as in the proof of Engeler’s Lemma.

We define a *hyperpath* to be a route in the hypertree from the root of  $\mathcal{T}[S, x]$  to a leaf. At a hypernode of the hypertree, the hyperpath goes through a specific channel ( $c_{ij}$  in Fig. 5). Similarly, a *reduced path* is a path in the reduced tree, ignoring the details of the hypernodes.

We exhibit a hyperpath in Fig. 6. This shows part of the hypertree. (Note that  $e_1, e_2, \dots$  here denote edges of the hypertree, not syntactic states.) To simplify the drawing, we ignore the details of the proc-subtrees, leaving only the enumerated channels of each hypernode.

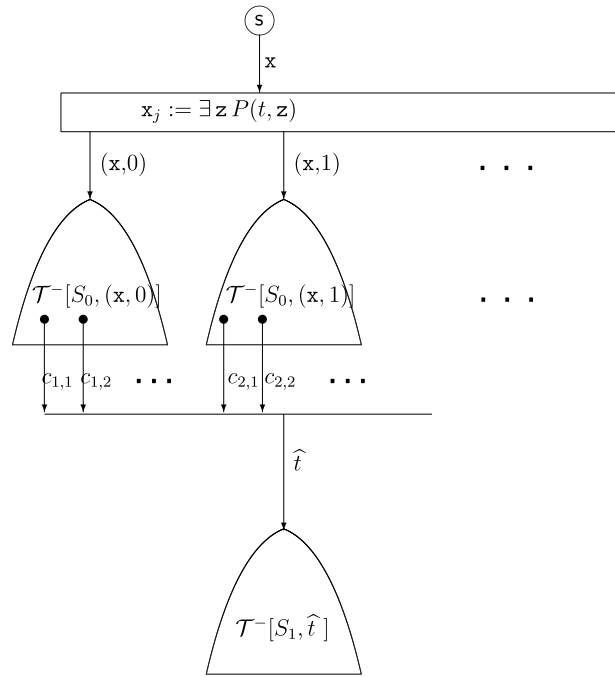


Fig. 5. Construction of hypertree (step 3).

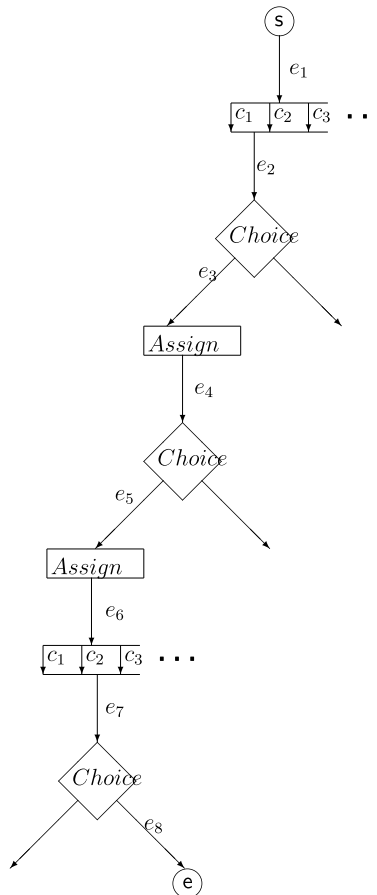


Fig. 6. Hyperpath.

From the root  $s$  to any given leaf  $e$  of the hypertree, there is one reduced path corresponding to infinitely many hyperpaths; for example, **hpath**(1, 2) consists of the edge  $e_1$ , the channel  $c_1$  of the 1st hypernode, the edges  $e_2, e_3, e_4, e_5, e_6$ , the channel  $c_2$  of the 2nd hypernode, and the edges  $e_7, e_8$ .

Notice that in an ‘Exist’ term  $\exists z P(t, z)$  in the tree, there may be other terms  $\exists z P'(t', z')$  inside the procedure  $P$ . Then (recursively) we expand all such proc-subtrees in  $P$  so as to form a hypertree without any ‘Exist’ nodes.

Notice also that on each hyperpath, there are three kinds of node:

- simple<sup>11</sup> assignment nodes  $x_j := t$ ,
- ‘Exist’ assignment nodes,
- branching nodes at boolean variables  $x^B$ .

Associated with each leaf are (infinitely many) hyperpaths, because of the multiple channels through the hypernodes which lead to that leaf.

We can then enumerate all the hyperpaths of each leaf as follows. Let **hpath**( $i_1, \dots, i_m$ ) be the hyperpath through the  $i_k$ -th channel at the  $k$ -th hypernode on the route (where  $k = 1, \dots, m$ , assuming there are  $m$  hypernodes on that hyperpath). Then we rename **hpath**( $i_1, \dots, i_m$ ) as **hpath**( $i$ ) where  $i = \ulcorner (i_1, \dots, i_m) \urcorner$ .

Finally, combining the enumeration of the leaves of the hypertree as above, and the enumeration of the hyperpaths of each leaf, we can effectively enumerate all hyperpaths of the computation tree.

Hence we have (compare Lemma 4.3.1):

**Lemma 4.5.1** (Engeler’s Lemma for **While**<sup>3N</sup>). *If a relation  $R$  is **While**<sup>3N</sup> semicomputable over a standard partial  $\Sigma$ -algebra  $A$ , then  $R$  can be expressed as the (strong) disjunction of an effective sequence of  $\Sigma^{\text{OR}}$ -booleans over  $A$ .*

**Proof.** Suppose  $R$  is the halting set in  $A$  of the **While**<sup>3N</sup> procedure

$$P \equiv \text{proc in } a \text{ out } b \text{ aux } c \text{ begin } S \text{ end.}$$

Consider the enumeration of the hyperpaths

$$\rho_0, \rho_1, \rho_2, \dots$$

of the hypertree for  $S$  as described above. For each hyperpath  $\rho$  (not leaf) of the computation tree  $\mathcal{T}[S, x]$  there is a boolean  $b_{S, \rho}$  with variables among  $x \equiv (a, b, c)$  which expresses the conjunction of results of the tests and the definability predicates from the root to the leaf of  $\mathcal{T}[S, x]$  along  $\rho$ . This boolean is constructed as follows.

A simple assignment node  $x := t$  in  $\rho$  contributes to  $b_{S, \rho}$  the conjunct

$$\dots \overset{c}{\wedge} \text{def}_s(t) \overset{c}{\wedge} \dots$$

which guarantees that  $b_{S, \rho}$  converges only if the evaluation of the term  $t$  converges at this point.

Suppose the hyperpath goes through a test node  $x^B$ . Consider the most recent assignment to  $x^B$  above this node. There are two cases:

- (1) The most recent assignment to  $x^B$  was a simple assignment  $x^B := t$ . Then we add either  $\dots \overset{c}{\wedge} x^B \overset{c}{\wedge} \dots$  or  $\dots \overset{c}{\wedge} \neg x^B \overset{c}{\wedge} \dots$  as a conjunct to  $b_{S, \rho}$ , according to whether the hyperpath goes to the left or right at this node.
- (2) The most recent assignment to  $x^B$  was an ‘Exist’ assignment  $x^B := \exists z P(t, z)$ . Then the hyperpath must go to the left (since in this case  $x^B$  must be true), and  $b_{S, \rho}$  is unchanged (since adding the conjunct true is redundant).

Define  $b_{S, k} \equiv b_{S, \rho_k}$ . Then  $R$  is expressed by the infinite disjunction

$$\bigvee_k b_{S, k} \tag{4.8}$$

just as in the proof of Engeler’s Lemma 4.3.1 for **While**<sup>(OR)</sup>.  $\square$

**Remark 4.5.2.** The sequence of booleans  $(b_{S, k})$  constructed in the above proof (4.8) does not, in general, satisfy semantic disjointedness (cf. Lemma 4.4.2), because of the nature of the **While**<sup>3N</sup> computation hypertree.

<sup>11</sup> Recall Remark 3.10.3(a).

## 5. Structure theorems for semicomputable sets over $\mathcal{R}$

In this section we present our structure theorems characterizing the **While**, **While**<sup>OR</sup> and **While**<sup>≡N</sup> semicomputable sets over  $\mathcal{R}$ . We will discuss the limitations of the **While** language in this regard, and show how the **While**<sup>OR</sup> and **While**<sup>≡N</sup> languages correct these deficiencies.

From now on, we will consider only the algebra  $A = \mathcal{R}$ , and write  $\Sigma$  for  $\Sigma(\mathcal{R})$ , and similarly for  $\Sigma^{\text{OR}}$  and  $\Sigma^{\text{≡N}}$ .

### 5.1. Computational equivalence; Semantics of atomic booleans

The proof of the Canonical Form [Lemma 5.2.2](#) below (and hence the Partition [Lemma 5.2.4](#)) requires a careful analysis of the semantics of atomic booleans of the forms (1)  $t_1 = t_2$  and (2)  $t_1 < t_2$ .

Assume for simplicity that  $t_1$  and  $t_2$  contain only the variable  $x$ : real.

These atomic booleans can be simplified, respectively, to the forms (1)  $p(x) = 0$  and (2)  $p(x) > 0$ , for some integer polynomial  $p(x)$ . Now according to the semantics of ‘=’ and ‘<’ ([Example 3.1.5](#)), together with the semantic rules for terms ([Definition 3.5.3](#)), the semantic evaluation of these two atoms at a state  $\sigma$ , where  $\sigma(x) = a$ , is given by

$$\llbracket p(x) = 0 \rrbracket \sigma \simeq \begin{cases} \uparrow & \text{if } p(a) = 0 \\ \mathbf{f} & \text{if } p(a) \neq 0 \end{cases} \quad (5.1a)$$

$$\llbracket p(x) > 0 \rrbracket \sigma \simeq \begin{cases} \mathbf{t} & \text{if } p(a) > 0 \\ \mathbf{f} & \text{if } p(a) < 0 \\ \uparrow & \text{if } p(a) = 0. \end{cases} \quad (5.1b)$$

Hence at a root  $a$  of  $p(x)$ , the booleans  $p(x) = 0$  and  $p(x) > 0$  both diverge.

Now suppose  $p(x)$  has degree 0, i.e.  $p(x) \equiv c$  for some (integer) constant  $c$ . Consider the two cases:

- (1)  $c \neq 0$ . Then  $p$  has no roots, and (as we would want) at all states  $p(x) = 0$  evaluates to  $\mathbf{f}$ , and  $p(x) > 0$  evaluates to  $\mathbf{t}$  if  $c$  is positive, and  $\mathbf{f}$  if  $c$  is negative.
- (2)  $c = 0$ . Now every real point is a root of  $p$ , but by (5.1) the atoms  $p(x) = 0$  and  $p(x) > 0$ , which simplify (resp.) to  $0 = 0$  and  $0 > 0$ , diverge at all states! But this is quite counter-intuitive.

Similarly, we would (presumably) want atoms  $t_1 = t_2$  to evaluate to  $\mathbf{t}$ , and not diverge, if (e.g.)  $t_1 \equiv t_2 \equiv 3$ , or  $t_1 \equiv 2 * x + 2$  and  $t_2 \equiv 1 + x + x + 1$ , or more generally, where the equality  $t_1 = t_2$  follows from the ring axioms over  $\mathbb{R}$ , and hence is true *a priori*.

Hence we must modify the semantics given by (5.1).

First, some remarks on representations of polynomials.

**Remark 5.1.1** (*Standard form for polynomials*). Any polynomial can be written in a standard form by (1) assuming a standard listing  $x_1, x_2, \dots$  of the real variables, and (2) ordering the monomials  $x_1^{e_1} \dots x_m^{e_m}$  (e.g.) first by decreasing weight ( $= e_1 + \dots + e_n$ ), and secondly, lexicographically in  $(e_1, \dots, e_n)$ , according to the order ‘>’ on  $\mathbb{N}$ . We can then define a coding of polynomials in standard form.

Note that our polynomial expressions in standard form have integer coefficients, although the signature  $\Sigma$  does not have a data type int. The point is that our “polynomial notation” does not involve integers essentially. For example, the polynomial expression ‘ $2x^2 - x + 3$ ’ stands for the  $\Sigma$ -term  $x * x + x * x + (-x) + 1 + 1 + 1$  (suitably parenthesized).

Now let  $\mathbf{E}$  be the equational calculus [[10](#), [§11.1](#)] in the language  $(0, 1, +, -, *,)$ , with the axioms for commutative rings with unit [[5](#)].

By “real term” we mean term of type real.

**Definition 5.1.2** (*Computational equivalence*). Two real terms  $t_1, t_2$  are *computationally equivalent* (written  $t_1 \cong t_2$ ) iff  $\mathbf{E} \vdash t_1 = t_2$ .

**Lemma 5.1.3**. Any real term  $t$  can be re-written uniquely as a polynomial in standard form; more precisely, there is a unique polynomial  $P[t]$  in standard form such that  $t \cong P[t]$ .

**Lemma 5.1.4**. For any two real terms  $t_1, t_2$ , the following three assertions are equivalent:

- (1)  $t_1 \cong t_2$ ,
- (2)  $P[t_1] \equiv P[t_2]$ ,
- (3)  $P[t_1 - t_2] \equiv 0$  (the zero polynomial).

Note that by the equivalence (1)  $\Leftrightarrow$  (2) above, computational equivalence between real terms is decidable.

The following lemma expresses the *soundness* and *completeness* of computational equivalence w.r.t. semantic equivalence.<sup>12</sup>

**Lemma 5.1.5.** For any two real terms  $t_1, t_2$

$$t_1 \cong t_2 \iff t_1 \approx t_2$$

**Proof.** ( $\Rightarrow$ ) is clear. ( $\Leftarrow$ ) follows from the fact that if a polynomial over  $\mathbb{R}$  has value 0 everywhere, then it must be the zero polynomial, by [Corollary 2.3.2](#).  $\square$

**Definition 5.1.6** (*Modified semantics of boolean atoms*). For real terms  $t_1, t_2$ , we define:

$$\llbracket t_1 = t_2 \rrbracket \sigma \simeq \begin{cases} \mathbf{t} & \text{if } t_1 \cong t_2 \\ \uparrow & \text{if } \llbracket t_1 \rrbracket \sigma = \llbracket t_2 \rrbracket \sigma \text{ but } t_1 \not\cong t_2 \\ \mathbf{f} & \text{if } \llbracket t_1 \rrbracket \sigma \neq \llbracket t_2 \rrbracket \sigma \end{cases} \quad (5.2a)$$

$$\llbracket t_1 < t_2 \rrbracket \sigma \simeq \begin{cases} \mathbf{t} & \text{if } \llbracket t_1 \rrbracket \sigma < \llbracket t_2 \rrbracket \sigma \\ \mathbf{f} & \text{if } \llbracket t_1 \rrbracket \sigma > \llbracket t_2 \rrbracket \sigma \text{ or } t_1 \cong t_2 \\ \uparrow & \text{if } \llbracket t_1 \rrbracket \sigma = \llbracket t_2 \rrbracket \sigma \text{ but } t_1 \not\cong t_2. \end{cases} \quad (5.2b)$$

These definitions will be used in the proof of the Canonical Form [Lemma 5.2.2](#).

**Discussion 5.1.7** (*Justification for modified semantics*). Again, as in [Discussion 3.1.7](#), we consider this issue in two ways: the first based on continuity considerations, and the second, again, based on a thought experiment involving concrete computations:

- (a) Recall [Discussion 3.1.7\(a\)](#) on the motivation for defining equality and order on the reals as partial functions  $\text{eq}_{\mathbb{R}}$  and  $\text{less}_{\mathbb{R}}$ . Continuity of **While**<sup>OR</sup> computable functions is a central concern here. We may then well ask: do the above modified semantics (5.2) not “spoil” this continuity result? The answer is no: with the above definitions, it still holds that **While** (or **While**<sup>OR</sup>, or **While**<sup>3N</sup>) computable functions are continuous. The proof depends on the fact that the condition for the atomic formula  $t_1 = t_2$  to have an output of  $\mathbf{t}$  instead of  $\uparrow$  (i.e. that  $t_1 \cong t_2$ ) is *independent* of the state. Hence the proof of the Continuity Theorem for **While** computable functions on topological algebras (see [Remark 3.1.4](#)), can be easily adapted to the present case, with the semantics based on [Definition 5.1.6](#). We omit details.
- (b) Another (“concrete”) approach to justifying this definition lies in continuing with our thought experiment in [Discussion 3.1.7\(b\)](#). So consider again an atomic formula of the form  $t_1 = t_2$ , and see what is involved in trying to decide whether it is true or not. First, take the case considered in [Discussion 3.1.7\(b\)](#) where  $t_1 \equiv x$ , and  $t_2 \equiv y$ . Suppose, again, that these are presented to us, at a given state  $\sigma$ , as fast Cauchy sequences  $(r_n)$  and  $(s_n)$  of rationals respectively. Then, as shown there ([Discussion 3.1.7\(b\)](#)), we can only gain “negative” information in finite time. In other words, if  $x = y$  is true at  $\sigma$ , then we cannot determine this in finite time, and so the computation diverges. Suppose, however, that (for example)  $t_1 \equiv 1 + x$  and  $t_2 \equiv x + 1$ . Then it is clear *a priori* that these terms are equal, regardless of the state, and without any need to consult the Cauchy sequence for  $x$  at that state. After all, it is the same variable, and hence the same Cauchy sequence, on both sides of the equation! Hence in this case we let the atom  $t_1 = t_2$  evaluate to  $\mathbf{t}$  at all states.

## 5.2. Canonical form for $\Sigma^{\text{OR}}$ booleans

Unless otherwise stated, the definitions and lemmas in this subsection refer to the  $\Sigma^{\text{OR}}$ -language, with the  $\Sigma$ -language as a special case. We generally write  $b, b', \dots$  for  $\Sigma^{\text{OR}}$ -booleans.

**Definition 5.2.1** (*Boolean combination*). A *boolean combination* of a set of atomic booleans is a boolean expression built up from the atoms  $t_1 < t_2$  and  $t_1 = t_2$  (with  $t_1, t_2 : \text{real}$ ) by  $\vee, \wedge, \overset{\text{c}}{\vee}, \overset{\text{c}}{\wedge}, \nabla, \Delta$  and  $\neg$ .

**Lemma 5.2.2** (*Canonical form for booleans over  $\mathcal{R}$* ). A  $\Sigma^{\text{OR}}$ -boolean with variables among  $x \equiv (x_1, \dots, x_n)$  of sort real only, is effectively semantically equivalent to a boolean combination of equations and inequalities of the form

$$p(x) = 0 \quad \text{and} \quad q(x) > 0$$

where  $p$  and  $q$  are polynomials in  $x$ .

<sup>12</sup> Recall [Definition 3.5.4](#) of semantic equivalence.

**Proof.** By structural induction on the boolean  $b$ .

Base cases:

- $b \equiv (t_1 = t_2)$  or  $(t_1 < t_2)$  for terms  $t_1, t_2 : \text{real}$ .  
By Lemma 5.1.4 these are semantically equivalent to (respectively)  $P[t_1 - t_2] = 0$  and  $P[t_2 - t_1] > 0$ .
- $b \equiv (t_1 = t_2)$  or  $(t_1 < t_2)$  for terms  $t_1, t_2 : \text{nat}$ . It is easy to see that every term  $t : \text{nat}$  without any nat variables must be closed, and in fact a numeral, i.e., of the form

$$\bar{n} \equiv \text{suc}(\text{suc}(\dots(\text{suc } 0)\dots)) \quad (n \text{ times 'suc'})$$

for some  $n \in \mathbb{N}$ . Hence in this case  $b$  has the form  $(\bar{n}_1 = \bar{n}_2)$  or  $(\bar{n}_1 < \bar{n}_2)$  for some  $n_1, n_2 \in \mathbb{N}$ , reducing to true or false in all cases.

*Induction step:* Suppose  $b_1$  and  $b_2$  are both effectively strongly equivalent to boolean combinations of equations and inequalities of the form  $p(x) = 0$  and  $q(x) > 0$ . Then clearly the same holds for  $\neg b, b_1 \vee b_2, b_1 \wedge b_2, b_1 \overset{c}{\vee} b_2, b_1 \overset{c}{\wedge} b_2, b_1 \nabla b_2$  and  $b_1 \triangle b_2$ .  $\square$

For a real variable  $x$ , let  $\mathbf{Bool}(x)$  be the set of  $\Sigma^{\text{OR}}$ -booleans with no free variables other than  $x$ . For the rest of this subsection, we consider only booleans in  $\mathbf{Bool}(x)$ .

**Definition 5.2.3.** For any  $b \in \mathbf{Bool}(x)$ , we define

- $\text{PS}(b)$  (the positive set of  $b$ ) =  $\{x \in \mathbb{R} \mid b[x] = \mathbf{t}\}$ .
- $\text{NS}(b)$  (the negative set of  $b$ ) =  $\{x \in \mathbb{R} \mid b[x] = \mathbf{f}\}$ .
- $\text{DS}(b)$  (the divergence set of  $b$ ) =  $\{x \in \mathbb{R} \mid b[x] \uparrow\}$ .

**Lemma 5.2.4** (Partition Lemma for booleans over  $\mathcal{R}$ ). Every boolean  $b \in \mathbf{Bool}(x)$  has semantics effectively represented by a partition of  $\mathbb{R}$  of the form:

$$\begin{aligned} \text{PS}(b) &= \bigcup_{i=1}^k I_i^+ \\ \text{NS}(b) &= \bigcup_{i=1}^\ell I_i^- \\ \text{DS}(b) &= \{d_1, \dots, d_m\} \end{aligned}$$

where  $k, \ell, m \geq 0$  and  $I_i^+, I_j^-$  are all disjoint algebraic open intervals, such that

$$\bigcup_{i=1}^k I_i^+ \cup \bigcup_{j=1}^\ell I_j^- \cup \{d_1, \dots, d_m\} = \mathbb{R}$$

and the divergence points  $d_1, \dots, d_m$  are precisely all the boundary points of  $b$ , i.e., the end points of the intervals  $I_1^+, \dots, I_k^+, I_1^-, \dots, I_\ell^-$ .

**Proof.** First convert the boolean to a canonical form given by the Canonical Form Lemma 5.2.2. We will then prove the lemma by structural induction on the boolean  $b$  in canonical form.

To clarify the details of the structural induction to follow, let us take a simple example: the case of two boolean  $b_1$  and  $b_2$  whose positive sets are single open intervals, i.e.  $\text{PS}(b_1) = (\alpha_1, \beta_1)$ , and  $\text{PS}(b_2) = (\alpha_2, \beta_2)$ , where (e.g.)  $\alpha_1 < \alpha_2 < \beta_1 < \beta_2$ . Then

$$\begin{aligned} \text{PS}(b_1 \vee b_2) &= (\alpha_1, \alpha_2) \cup (\alpha_2, \beta_1) \cup (\beta_1, \beta_2) \\ \text{PS}(b_1 \overset{c}{\vee} b_2) &= (\alpha_1, \beta_1) \cup (\beta_1, \beta_2) \\ \text{PS}(b_1 \nabla b_2) &= (\alpha_1, \beta_2). \end{aligned}$$

We now proceed by structural induction on  $b$ .

Base case:  $b \equiv p(x) = 0$  or  $p(x) > 0$ . Use Corollary 2.3.4.

Note that in the case that  $p(x)$  has degree 0, i.e., it is a constant integer  $c$ , the atomic boolean  $p(x) > 0$  has the form  $c > 0$ , and so reduces to true or false, depending on the value of  $c$ . Similarly with the case of an atomic boolean  $p(x) = 0$ .

*Induction step.* Briefly, this follows from the fact that the class of finite unions and intersection of algebraic intervals is closed under (binary) union and intersection. In more detail, we consider the various cases:

- $b \equiv -b_1$ . Just interchange the positive and negative sets for  $b$  and  $b_1$ .

Next, suppose:

$$\text{PS}(b_1) = \bigcup_{i=1}^{k_1} I_{1i}^+$$

$$\text{NS}(b_1) = \bigcup_{i=1}^{\ell_1} I_{1i}^-$$

$$\text{DS}(b_1) = \{d_{11}, \dots, d_{1m_1}\}$$

$$\text{PS}(b_2) = \bigcup_{j=1}^{k_2} I_{2j}^+$$

$$\text{NS}(b_2) = \bigcup_{j=1}^{\ell_2} I_{2j}^-$$

$$\text{DS}(b_2) = \{d_{21}, \dots, d_{2m_2}\}.$$

- $b \equiv b_1 \vee b_2$ . Then

$$\text{PS}(b) = \left( \bigcup_{i=1}^{k_1} \bigcup_{j=1}^{k_2} (I_{1i}^+ \cap I_{2j}^+) \right) \cup \left( \bigcup_{i=1}^{k_1} \bigcup_{j=1}^{\ell_2} (I_{1i}^+ \cap I_{2j}^-) \right) \cup \left( \bigcup_{i=1}^{\ell_1} \bigcup_{j=1}^{k_2} (I_{1i}^- \cap I_{2j}^+) \right)$$

$$\text{NS}(b) = \bigcup_{i=1}^{\ell_1} \bigcup_{j=1}^{\ell_2} (I_{1i}^- \cap I_{2j}^-)$$

$$\text{DS}(b) = \{d_{11}, \dots, d_{1m_1}, d_{21}, \dots, d_{2m_2}\}.$$

- $b \equiv b_1 \wedge b_2$ . Then

$$\text{PS}(b) = \bigcup_{i=1}^{k_1} \bigcup_{j=1}^{k_2} (I_{1i}^+ \cap I_{2j}^+)$$

$$\text{NS}(b) = \left( \bigcup_{i=1}^{\ell_1} \bigcup_{j=1}^{\ell_2} (I_{1i}^- \cap I_{2j}^-) \right) \cup \left( \bigcup_{i=1}^{k_1} \bigcup_{j=1}^{\ell_2} (I_{1i}^+ \cap I_{2j}^-) \right) \cup \left( \bigcup_{i=1}^{\ell_1} \bigcup_{j=1}^{k_2} (I_{1i}^- \cap I_{2j}^+) \right)$$

$$\text{DS}(b) = \{d_{11}, \dots, d_{1m_1}, d_{21}, \dots, d_{2m_2}\}.$$

- $b \equiv b_1 \overset{\circ}{\vee} b_2$ . Then

$$\text{PS}(b) = \bigcup_{i=1}^{k_1} \bigcup_{j=1}^{\ell_1} \bigcup_{k=1}^{k_2} (I_{1i}^+ \cup (I_{1j}^- \cap I_{2k}^+))$$

$$\text{NS}(b) = \bigcup_{i=1}^{\ell_1} \bigcup_{j=1}^{\ell_2} (I_{1i}^- \cap I_{2j}^-)$$

$$\text{DS}(b) = \{d_{11}, \dots, d_{1m_1}\} \cup \{d_{2j} \mid \exists i: d_{2j} \in I_{1i}^-\}.$$

- $b \equiv b_1 \overset{\circ}{\wedge} b_2$ . Then

$$\text{PS}(b) = \bigcup_{i=1}^{k_1} \bigcup_{j=1}^{k_2} (I_{1i}^+ \cap I_{2j}^+)$$

$$\text{NS}(b) = \bigcup_{i=1}^{\ell_1} \bigcup_{j=1}^{m_1} \bigcup_{k=1}^{\ell_2} (I_{1i}^- \cup (I_{1j}^+ \cap I_{2k}^-))$$

$$\text{DS}(b) = \{d_{11}, \dots, d_{1m_1}\} \cup \{d_{2j} \mid \exists i: d_{2j} \in I_{1i}^+\}.$$

- $b \equiv b_1 \nabla b_2$ . Then

$$\text{PS}(b) = \bigcup_{i=1}^{k_1} \bigcup_{j=1}^{k_2} (I_{1i}^+ \cup I_{2j}^+)$$

$$\text{NS}(b) = \bigcup_{i=1}^{\ell_1} \bigcup_{j=1}^{\ell_2} (I_{1i}^- \cap I_{2j}^-)$$

$$\text{DS}(b) = (\{d_{11}, \dots, d_{1m_1}\} \cap \{d_{21}, \dots, d_{2m_2}\}) \cup \{d_{1i} \mid \exists j: d_{1i} \in I_{2j}^-\} \cup \{d_{2i} \mid \exists j: d_{2i} \in I_{1j}^-\}.$$

- $b \equiv b_1 \triangle b_2$ . Then

$$\text{PS}(b) = \bigcup_{i=1}^{k_1} \bigcup_{j=1}^{k_2} (I_{1i}^+ \cap I_{2j}^+)$$

$$\text{NS}(b) = \bigcup_{i=1}^{\ell_1} \bigcup_{j=1}^{\ell_2} (I_{1i}^- \cup I_{2j}^-)$$

$$\text{DS}(b) = (\{d_{11}, \dots, d_{1m_1}\} \cap \{d_{21}, \dots, d_{2m_2}\}) \cup \{d_{1i} \mid \exists j: d_{1i} \in I_{2j}^+\} \cup \{d_{2i} \mid \exists j: d_{2i} \in I_{1j}^+\}. \quad \square$$

Next we give several lemmas in preparation for the structure theorems in Section 5.5.

**Lemma 5.2.5.** *There is a **While** computable embedding*

$$\iota_{\mathbb{N}}: \mathbb{N} \hookrightarrow \mathbb{R}.$$

**Proof.** By a simple while loop.  $\square$

From this we easily get:

**Lemma 5.2.6.** *There is a **While** computable injection*

$$\iota_{\mathbb{Z}}: \mathbb{N} \rightarrow \mathbb{R}$$

where for any  $m \in \mathbb{Z}$ ,  $\iota_{\mathbb{Z}}(\ulcorner m \urcorner) = m \in \mathbb{R}$ .

**Lemma 5.2.7.** *There is a **While** computable function*

$$\mathit{eval}: \mathbb{N} \times \mathbb{R} \rightarrow \mathbb{R}$$

such that for any integer polynomial  $p$  and  $a \in \mathbb{R}$ :

$$\mathit{eval}(\ulcorner p \urcorner, a) = p(a).$$

**Proof.** The function  $\mathit{eval}$  is defined by induction on the degree of  $p$ . We omit details.  $\square$

Note that this lemma is a special case of the *term evaluation property* (TEP) for  $\mathcal{R}$  [14, §4.7]. In fact, it is the main step in proving the TEP for  $\mathcal{R}$ .

**Lemma 5.2.8.** *There is a **While** computable function*

$$\mathit{less}_{\mathbb{Q}}: \mathbb{N} \times \mathbb{R} \rightarrow \mathbb{B}$$

such that for  $r \in \mathbb{Q}$  and  $x \in \mathbb{R}$ :

$$\mathit{less}_{\mathbb{Q}}(\ulcorner r \urcorner, x) \simeq \begin{cases} \mathbf{t} & \text{if } r < x \\ \mathbf{f} & \text{if } r > x \\ \uparrow & \text{if } r = x. \end{cases}$$



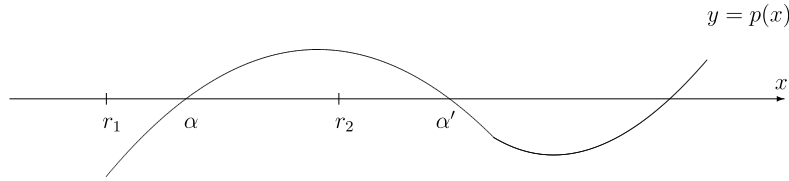


Fig. 7. Proof of Lemma 5.2.11, case 1.

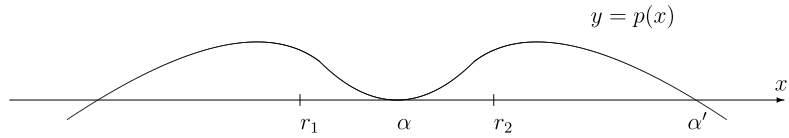


Fig. 8. Proof of Lemma 5.2.11, case 3.

**Proof.** Let  $r = \frac{m}{n+1}$  ( $m \in \mathbb{Z}, n \in \mathbb{N}$ ). Then by our assumptions on the coding, we can primitive recursively retrieve  $\ulcorner m \urcorner$  and  $n$  from  $\ulcorner r \urcorner$ . Then (using Lemmas 5.2.5 and 5.2.6) define

$$\mathbf{less}_Q(\ulcorner r \urcorner, x) \iff_{df} \mathbf{t}_Z(\ulcorner m \urcorner) < (\mathbf{t}_N(n) + 1) \times x. \quad \square$$

**Lemma 5.2.9.** *There is a **While**<sup>OR</sup> computable function*

$$\mathbf{less}_A : \mathbb{N} \times \mathbb{R} \rightarrow \mathbb{B}$$

such that for  $\alpha \in \mathbb{A}$  and  $x \in \mathbb{R}$ :

$$\mathbf{less}_A(\ulcorner \alpha \urcorner, x) \simeq \begin{cases} \mathbf{t} & \text{if } \alpha < x \\ \mathbf{f} & \text{if } \alpha > x \\ \uparrow & \text{if } \alpha = x. \end{cases}$$

**Proof.** We can effectively retrieve from  $\ulcorner \alpha \urcorner$  the numbers  $\ulcorner p \urcorner$  and  $k$ , where  $\alpha$  is the  $k$ -th root of  $p$ .

Then by Lemma 5.2.7, we have a **While** computable function **eval** such that  $\mathbf{eval}(\ulcorner p \urcorner, a) = p(a)$ . Also by Lemma 2.3.5, we can effectively find two rationals  $r_1$  and  $r_2$  such that  $r_1 < \alpha < r_2$  and  $\alpha$  is the only root of  $p$  between these two rationals.

There are now four cases, which can be effectively distinguished by Sturm’s Theorem.

In cases 1 and 2,  $\alpha$  is a single root. In case 1 (Fig. 7)  $p(x)$  changes sign from negative to positive at  $\alpha$ . Here we can see that  $\alpha < x$  iff

$$[(r_1 < x) \wedge (p(x) > 0)] \nabla [r_2 < x].$$

Note the use of the strong disjunction. (See Remark 5.2.10 below.)

In case 2 (formed by reflecting Fig. 7 about the  $x$  axis)  $p(x)$  changes sign from positive to negative. This reduces to case 1 by replacing  $p(x)$  by  $-p(x)$ .

In cases 3 and 4,  $\alpha$  is a repeated root. In case 3 (Fig. 8)  $p(x)$  is positive near  $\alpha$ . Here, by choosing  $r_1$  and  $r_2$  sufficiently close to  $\alpha$  (so that  $p'(r_2) > 0$ ,  $p'(r_1) < 0$  and there is no root of  $p'(x)$  between  $r_1$  and  $\alpha$ , or between  $\alpha$  and  $r_2$ ) we have  $\alpha < x$  iff

$$[(r_1 < x) \wedge (p'(x) > 0)] \nabla [r_2 < x]$$

where  $p'$  is the derivative of  $p$ .

Note again the use of the strong disjunction here.

In case 4 (formed by reflecting Fig. 8 about the  $x$  axis)  $p(x)$  is negative near  $\alpha$ . This reduces to case 3 by replacing  $p(x)$  by  $-p(x)$ .

Note that all the above operations on polynomials are effective; for example,  $\ulcorner -p \urcorner$  and  $\ulcorner p' \urcorner$  are primitive recursive in  $\ulcorner p \urcorner$ .  $\square$

**Remark 5.2.10 (Need for strong disjunction).** In case 1, if  $x = r_2$ , then the disjunct  $(r_2 < x)$  will diverge, and so we need ‘ $\nabla$ ’ to make the whole expression converge. Similarly for the other cases.

**Lemma 5.2.11.** *There is a **While** computable function*

$$\mathbf{in}_Q : \mathbb{N} \times \mathbb{R} \rightarrow \mathbb{B}$$

such that

$$\mathbf{in}_Q(\ulcorner(r_1, r_2)\urcorner, x) \simeq \begin{cases} \mathbf{t} & \text{if } x \in (r_1, r_2) \\ \mathbf{f} & \text{if } x < r_1 \text{ or } r_2 < x \\ \uparrow & \text{otherwise, i.e. if } x = r_1 \text{ or } x = r_2 \end{cases}$$

where  $r_1$  and  $r_2$  are rationals with  $r_1 < r_2$ .

**Proof.** We can primitively recursively retrieve  $\ulcorner r_1 \urcorner$  and  $\ulcorner r_2 \urcorner$  from  $\ulcorner(r_1, r_2)\urcorner$ , and therefore define:

$$\mathbf{in}_Q(\ulcorner(r_1, r_2)\urcorner, x) \iff_{df} \mathbf{less}_Q(\ulcorner r_1 \urcorner, x) \wedge \neg \mathbf{less}_Q(\ulcorner r_2 \urcorner, x)$$

which is **While** computable, by [Lemma 5.2.8](#).  $\square$

**Lemma 5.2.12.** There is a **While**<sup>OR</sup> computable function

$$\mathbf{in}_A : \mathbb{N} \times \mathbb{R} \rightarrow \mathbb{B}$$

such that

$$\mathbf{in}_A(\ulcorner(\alpha, \beta)\urcorner, x) \simeq \begin{cases} \mathbf{t} & \text{if } x \in (\alpha, \beta) \\ \mathbf{f} & \text{if } x < \alpha \text{ or } \beta < x \\ \uparrow & \text{otherwise} \end{cases}$$

where  $\alpha$  and  $\beta$  are algebraic numbers with  $\alpha < \beta$ .

**Proof.** Like the proof of [Lemma 5.2.11](#), except that instead of  $\mathbf{less}_Q$ , we use  $\mathbf{less}_A$ , which is **While**<sup>OR</sup> computable by [Lemma 5.2.9](#).  $\square$

### 5.3. Characterizations of semicomputable real sets

In this subsection, we prove the ‘ $\Rightarrow$ ’ direction of the structure theorems given in [Section 5.5](#) below.

**Lemma 5.3.1.** If a set  $R \subseteq \mathbb{R}$  is **While**<sup>OR</sup> semicomputable over  $\mathcal{R}$ , then  $R$  is the union of an effective countable sequence of disjoint algebraic intervals.

**Proof.** If  $R \subseteq \mathbb{R}$  is **While**<sup>OR</sup> semicomputable, then by Engeler’s [Lemma 4.3.1](#) for the **While**<sup>OR</sup> language,

$$a \in R \iff \bigvee_{k=0}^{\infty} b_k[a]$$

for an effective sequence  $(b_k)$  of  $\Sigma^{\text{OR}}$ -booleans in **Bool**( $x$ ). By the Semantic Disjointness [Lemma 4.4.2](#), this sequence  $(b_k)$  is semantically disjoint, and, further, by the Partition [Lemma 5.2.4](#), each  $b_k$  itself defines an effective finite union of disjoint algebraic intervals.  $\square$

**Lemma 5.3.2.** If  $R \subseteq \mathbb{R}$  is **While**<sup>OR</sup> semicomputable over  $\mathcal{R}$ , then  $R$  is the union of an effective countable sequence of rational intervals.

**Proof.** By [Lemmas 5.3.1 and 2.2.5](#).  $\square$

**Remark 5.3.3.** We lose disjointness here, because the rational intervals generated by the proof of [Lemma 2.2.5](#) are not disjoint.

**Lemma 5.3.4.** If  $R \subseteq \mathbb{R}$  is **While**<sup>3N</sup> semicomputable over  $\mathcal{R}$ , then  $R$  is the union of an effective countable sequence of algebraic intervals.

**Proof.** By Engeler’s [Lemma 4.5.1](#) for **While**<sup>3N</sup>, a **While**<sup>3N</sup> semicomputable set over  $\mathcal{R}$  can be expressed as a countable disjunction of an effective sequence of  $\Sigma^{\text{OR}}$ -booleans, to which the Partition Lemma again applies.  $\square$

Note again the lack of disjointness of the sequence of intervals obtained here. (See [Remark 4.5.2](#).)

**Lemma 5.3.5.** If  $R \subseteq \mathbb{R}$  is **While**<sup>3N</sup> semicomputable over  $\mathcal{R}$ , then  $R$  is the union of an effective countable sequence of rational intervals.

**Proof.** By [Lemmas 5.3.4 and 2.2.5](#).  $\square$

Note that we could have proved [Lemma 5.3.2](#) as an immediate consequences of this lemma.

#### 5.4. Unions of eff. sequences of intervals are semicomputable

We will now prove the reverse ' $\Leftarrow$ ' direction of the structure theorems.

**Lemma 5.4.1.** *The union of an effective countable sequence of disjoint rational intervals is **While** semicomputable over  $\mathcal{R}$ .*

**Proof.** An effective sequence of rational intervals gives us a total recursive function  $f : \mathbb{N} \rightarrow \mathbb{N}$  such that  $f(n)$  is the code of the  $n$ -th rational interval. So the union of such a sequence of intervals is the halting set of the procedure

```

proc
in  x : real;
aux i : nat;
begin
  i := 0;
  while not(inQ(Pf(i), x))
  do i := i + 1 od
end

```

where  $P_f$  is the **While**( $\mathcal{N}$ ) (and hence **While**( $\mathcal{R}$ )) procedure which computes  $f$ .

By Lemma 5.2.11, **in**<sub>Q</sub> is **While** computable, and so the above procedure is **While** computable.  $\square$

#### Remarks 5.4.2.

- (a) In the above procedure, if  $x$  lies on the boundary of one of the intervals, there will be “local divergence”. But in that case, by the *disjointness assumption*,  $x$  cannot lie in any of the other intervals, so this divergence still gives the correct result.
- (b) This result is related to Lemma 4.4.3, which states that a disjunction of an effective sequence of semantically disjoint booleans can be evaluated “from the left”, i.e., by a ‘while’ loop.

**Lemma 5.4.3.** *The union of an effective countable sequence of disjoint algebraic intervals is **While**<sup>OR</sup> semicomputable over  $\mathcal{R}$ .*

**Proof.** Just like the previous Lemma, but instead of **in**<sub>Q</sub>( $f(i), r$ ), we use **in**<sub>A</sub>( $P_f(i), x$ ) which is **While**<sup>OR</sup> computable, by Lemma 5.2.12.  $\square$

**Lemma 5.4.4.** *The union of an effective countable sequence of algebraic intervals is **While**<sup>3N</sup> semicomputable over  $\mathcal{R}$ .*

**Proof.** An effective sequence of algebraic intervals is given by a total **While** computable function  $f : \mathbb{N} \rightarrow \mathbb{N}$  such that  $f(n)$  returns the code of the  $n$ -th algebraic interval.

So the countable union of an effective sequence of algebraic intervals is the halting set of the following **While**<sup>3N</sup> procedure:

```

proc
in  x : real;
out b : bool;
begin
  b :=  $\exists z P(x, z)$ 
end

```

where  $P(x, z)$  is the procedure defined as

```

proc
in  x : real;
    z : nat;
out b : bool;
begin
  b := inA(Pf(z), x);
end

```

and  $P_f : \text{nat} \rightarrow \text{nat}$  is the **While**( $\mathcal{N}$ ) (and hence **While**( $\mathcal{R}$ )) procedure which computes  $f$ .

By Lemma 5.2.12, **in**<sub>A</sub> is **While**<sup>OR</sup> (and hence **While**<sup>3N</sup>) computable, and so the above procedure is **While**<sup>3N</sup> computable.  $\square$

**Corollary 5.4.5.** *The union of an effective countable sequence of rational intervals is  $\mathbf{While}^{\exists\mathbb{N}}$  semicomputable over  $\mathcal{R}$ .*

**Proof.** By Lemma 2.2.4, an effective sequence of rational intervals is also an effective sequence of algebraic intervals.  $\square$

**Discussion 5.4.6.** To summarize the results of this subsection: When an effective sequence of rational (or algebraic) intervals is disjoint, we can represent their union as the halting set of a  $\mathbf{While}$  (or  $\mathbf{While}^{\text{OR}}$ , respectively) procedure (as in Lemmas 5.4.1 and 5.4.3), since it can be evaluated from the left, i.e., by a ‘while’ loop.

However when the intervals are not disjoint, their union must be evaluated by a  $\mathbf{While}^{\exists\mathbb{N}}$  procedure, using the ‘Exist’ construct (as in Lemma 5.4.4 and Corollary 5.4.5).

### 5.5. Structure theorems for semicomputable sets over $\mathcal{R}$

We present our three structure theorems for  $\mathbf{While}^{\text{OR}}$  and  $\mathbf{While}^{\exists\mathbb{N}}$  semicomputable sets over  $\mathcal{R}$ .

**Theorem 1.** *A subset of  $\mathbb{R}$  is  $\mathbf{While}^{\text{OR}}$  semicomputable over  $\mathcal{R}$  iff it is the union of an effective countable sequence of disjoint algebraic intervals.*

**Proof.** By Lemmas 5.3.1 and 5.4.3.  $\square$

**Theorem 2.** *A subset of  $\mathbb{R}$  is  $\mathbf{While}^{\exists\mathbb{N}}$  semicomputable over  $\mathcal{R}$  iff it is the union of an effective countable sequence of algebraic intervals.*

**Proof.** By Lemmas 5.3.4 and 5.4.4.  $\square$

**Theorem 3.** *A subset of  $\mathbb{R}$  is  $\mathbf{While}^{\exists\mathbb{N}}$  semicomputable over  $\mathcal{R}$  iff it is the union of an effective countable sequence of rational intervals.*

**Proof.** By Lemma 5.3.5 and Corollary 5.4.5.  $\square$

We do not have a structure theorem for  $\mathbf{While}$  semicomputable sets. We only have a partial result:

**Theorem 4.** *For subsets of  $\mathbb{R}$ ,*

- (a)  $\mathbf{While}$  semicomputable over  $\mathcal{R} \implies$  union of effective sequence of rational intervals.
- (b) Union of effective sequence of disjoint rational intervals  $\implies \mathbf{While}$  semicomputable over  $\mathcal{R}$ .

**Proof.** (a) By Lemma 5.3.2.

(b) By Lemma 5.4.1.  $\square$

See Remarks 5.3.3 and 5.4.2(a) for the reasons that disjointedness is lost in part (a), but needed in part (b).

### 5.6. Projectively $\mathbf{While}^{\exists\mathbb{N}}$ semicomputable sets

We now prove that for the  $\mathbf{While}^{\exists\mathbb{N}}$  language, projective semicomputability is equivalent to semicomputability, i.e., semicomputability is closed under projection onto  $\mathbb{R}$ .

**Lemma 5.6.1.** *Given a continuous partial function  $b : \mathbb{R}^n \rightharpoonup \mathbb{B}$ , if there exists an  $n$ -tuple of reals  $x = (x_1, \dots, x_n)$  such that  $b(x) \downarrow \mathbf{t}$ , then there exists an  $n$ -tuple of rationals  $r = (r_1, \dots, r_n)$  such that  $b(r) \downarrow \mathbf{t}$ .*

**Proof.** Suppose there exists a real tuple  $x = (x_1, \dots, x_n) \in \mathbb{R}^n$  such that  $b(x) \downarrow \mathbf{t}$ . Then by continuity of  $b$ , there exists  $\delta > 0$  such that for all real tuples  $y = (y_1, \dots, y_n)$  in the neighborhood set

$$N(x, \delta) =_{df} \{ (y_1, \dots, y_n) \mid \sqrt{(x_1 - y_1)^2 + \dots + (x_n - y_n)^2} < \delta \},$$

we have  $b(y) \downarrow \mathbf{t}$ . But then, because of the density of  $\mathbb{Q}$  in  $\mathbb{R}$ , there exists a rational tuple  $r = (r_1, \dots, r_n) \in N(x, \delta)$ .  $\square$

**Theorem 5.** *A set  $R \subseteq \mathbb{R}^n$  is  $\mathbf{While}^{\exists\mathbb{N}}$  projectively semicomputable over  $\mathcal{R}$  if, and only if,  $R$  is  $\mathbf{While}^{\exists\mathbb{N}}$  semicomputable over  $\mathcal{R}$ .*

**Proof.** ‘ $\Leftarrow$ ’: Trivial.

' $\Rightarrow$ ': Suppose  $R \subseteq \mathbb{R}^n$  is **While**<sup>∃N</sup> projectively semicomputable. Then<sup>13</sup> there exists a **While**<sup>∃N</sup> semicomputable relation  $R' \subset \mathbb{R}^{m+n}$ , such that for all  $x \in \mathbb{R}^n$ :

$$\begin{aligned} x \in R &\iff \exists y \in \mathbb{R}^m (x, y) \in R' \\ &\iff \exists y \in \mathbb{R}^m \bigvee_k b_k[x, y] \\ &\quad \text{for some effective sequence } (b_k) \text{ of } \Sigma^{\text{OR}}\text{-booleans,} \\ &\quad \text{by Engeler's Lemma 4.5.1 for } \mathbf{While}^{\exists N} \text{ applied to } R' \\ &\iff \bigvee_k \exists y \in \mathbb{R}^m b_k[x, y] \\ &\iff \bigvee_k \exists r \in \mathbb{Q}^m b_k[x, r], \quad \text{by Lemma 5.6.1.} \end{aligned}$$

It is not hard to see that we can construct an effective double sequence  $(b_{k,\ell})$  of  $\Sigma^{\text{OR}}$ -booleans, such that for all  $k, \ell$ , if  $\ell = \lceil r \rceil$  then for all  $x \in \mathbb{R}^n$

$$\begin{aligned} b_{k,\ell}[x] &\iff b_k[x, r] \quad \text{and so} \\ x \in R &\iff \bigvee_k \bigvee_\ell b_{k,\ell}[x]. \end{aligned} \tag{5.3}$$

Finally, by a method similar to that in the proof of Lemma 5.4.4, we can show that the r.h.s. of (5.3) is the halting set of a **While**<sup>∃N</sup> procedure.  $\square$

Essentially, the above proof involves replacing existential quantification over  $\mathbb{R}$  by existential quantification over  $\mathbb{Q}$  (using continuity and density of  $\mathbb{Q}$  in  $\mathbb{R}$ ), and then replacing the latter by a countable disjunction.

### Remarks 5.6.2.

- (a) We do not know if this result holds for **While** or **While**<sup>OR</sup>.
- (b) In a total (non-topological) algebra  $\mathcal{R}_t$  over the reals, the continuity argument in the above proof would not work, and in fact, the theorem fails! A counterexample is given in [14, §6.2].

## 6. Conclusion and future work

### 6.1. Conclusion

We have investigated computability, or rather semicomputability, for the **While** language and certain extensions (**While**<sup>OR</sup> and **While**<sup>∃N</sup>) over a topological partial algebra  $\mathcal{R}$  on the reals. We proved four structure theorems for semicomputable sets in  $\mathcal{R}$ , of the form: a subset of  $\mathbb{R}$  is semicomputable in the **While** language, or one of these extensions, if, and only if, it is the union of an effective countable sequence of rational (or algebraic) open intervals (see Section 5.5).

We also proved a fifth theorem, stating that in  $\mathcal{R}$ , projective **While**<sup>∃N</sup> semicomputability is equivalent to **While**<sup>∃N</sup> semicomputability.

### 6.2. Future work and conjectures

We list some ideas for future work in this area, and conjectures:

- (1) Expanding  $\mathcal{R}$  by including division by naturals:

$$\begin{aligned} \text{div}^N : \mathbb{R} \times \mathbb{N} &\rightarrow \mathbb{R}, \quad \text{where} \\ \text{div}^N(x, n) &= \frac{x}{n+1} \end{aligned}$$

(This is easily seen to be equivalent to the expansion of  $\mathcal{R}$  formed by adding multiplication of a real by a rational.)

Now we can directly embed  $\mathbb{Q}$  in  $\mathbb{R}$ . The Canonical Form and Partition Lemmas still hold. In fact it seems clear that the five theorems in Section 5.5 also hold for the algebra  $\mathcal{R} + \text{div}^N$ .

<sup>13</sup> The ' $\bigvee_k$ ' symbol below indicates strong disjunction (Section 4.2(2)).

(2) Expanding  $\mathcal{R}$  to an algebra  $\mathcal{R}^{\text{div}}$ , which includes the (partial) real division operation

$$\begin{aligned} \text{div} : \mathbb{R}^2 &\rightarrow \mathbb{R}, \quad \text{where} \\ \text{div}(x, y) &\simeq \frac{x}{y}. \end{aligned}$$

This expansion is a major step compared to (1). The Canonical Form Lemma now becomes:

A  $\Sigma^{\text{OR}}$ -boolean over  $\mathcal{R}^{\text{div}}$  is effectively semantically equivalent to a boolean combination of equations and inequalities of the form  $r(x) = 0$  and  $r(x) > 0$ , where

$$r(x) \equiv \frac{p(x)}{q(x)} \tag{6.1}$$

with  $p(x), q(x)$  integer polynomials.

The important thing to notice is that the zeros and poles of rational functions are algebraic numbers, since in Eq. (6.1) the zeros and poles of  $r(x)$  are respectively the roots of  $p(x)$  and of  $q(x)$ . Thus it can be seen that the Partition Lemma still holds for  $\mathcal{R}^{\text{div}}$ . In fact:

$$\frac{p(x)}{q(x)} > 0 \iff (p(x) \times q(x)) > 0$$

where “ $\iff$ ” is weak semantic equivalence.<sup>14</sup>

We conjecture that the four structure theorems in Section 5.5 hold for  $\mathcal{R}^{\text{div}}$ .

(3) Investigating the structure of semicomputable subsets of  $\mathbb{R}^m$  for  $m > 1$ . Although the Canonical Form Lemma 5.2.2 for booleans over  $\mathcal{R}$  still holds when  $m > 1$ , the Partition Lemma 5.2.4 for booleans is problematic – even the formulation of a suitable generalization of it to  $m > 1$  provides a challenge. A useful approach may be the method of cell decomposition, applied to o-minimal structures on  $\mathbb{R}$  [18].<sup>15</sup>

(4) Bridging the gap between abstract models (e.g. **While**<sup>∃N</sup>) and concrete models of computation over  $\mathbb{R}$  (e.g. Weihrauch’s TTE [20]).

We have seen (Theorem 3) that for a relation  $R$  on  $\mathbb{R}$ :

$$R \text{ is } \mathbf{While}^{\exists N} \text{ semicomputable in } \mathcal{R} \iff R = \bigcup_k I_k \tag{6.2}$$

where  $(I_k)$  is an effective sequence of rational intervals.

On the other hand, Weihrauch has shown [20] that for his concrete model:

$$R \text{ is TTE-semicomputable} \iff R = \bigcap_j \bigcup_k I_{j,k} \tag{6.3}$$

where  $(I_{j,k})$  is an effective double sequence of rational intervals.

We can try to bridge the gap between (6.2) and (6.3) by generalizing the notion of semicomputability in  $\mathcal{R}$  to that of approximable **While**<sup>∃N</sup> semicomputability, where a set  $R \subseteq \mathbb{R}^n$  is said to be approximably **While**<sup>∃N</sup> semicomputable if for some **While**<sup>∃N</sup> procedure

$$\begin{aligned} &P : \text{nat} \times \text{real} \rightarrow \text{bool} \\ \text{writing } &P_n^{\mathcal{R}}(x) =_{df} P^{\mathcal{R}}(n, x) \\ \text{we have } &R = \bigcap_n \mathbf{Halt}^{\mathcal{R}}(P_n^{\mathcal{R}}). \end{aligned}$$

We then conjecture that for a set  $R \subseteq \mathbb{R}^m$ :

$$R \text{ is approx. } \mathbf{While}^{\exists N} \text{ semicomp.} \iff R \text{ is TTE semicomp.}$$

The motivation for this conjecture, and the reason for the terminology “approximable semicomputability”, is by analogy with the “completeness theorem” in [15], where for partial topological algebras  $A$  (such as  $\mathcal{R}$ ) satisfying certain general conditions, it was proved that

$$\mathbf{WhileCC} \text{ approx. computability} \iff \text{concrete computability.}$$

<sup>14</sup> Recall Definition 3.5.5.

<sup>15</sup> We thank Patrick Speissegger for this suggestion.

Here **WhileCC** is the **While** language extended by a nondeterministic “countable choice” operator (see [Remark 3.9.2](#)), and a function  $f: A^u \rightarrow A^v$  is said to be *approximately WhileCC computable* if for some **WhileCC** procedure  $P: \text{nat} \times u \rightarrow v$ , the sequence of (many-valued) functions

$$P_n^A: A^u \rightarrow A^v$$

converges or *approximates* to  $f$  (in a suitable sense).

In other words, for *abstract computability* to correspond to *concrete computability*, it must be augmented by

- (a) a nondeterministic choice operator ‘choose’ on  $\mathbb{N}$ ,
- (b) approximability of computations.

Similarly, in the present case, we conjecture that for *abstract semicomputability* to correspond to *concrete semicomputability*, it must be augmented by

- (a) the ‘Exist’ operator on  $\mathbb{N}$ ,
- (b) approximability, which here means taking countable intersections.

Note that our ‘Exist’ operator can be viewed as a “weakly deterministic” special case of the ‘choose’ operator (see again [Remark 3.9.2](#)).

### Acknowledgements

This paper developed out of the first author’s M.Sc. Thesis [21]. We are grateful to Jacques Carette, Patrick Speissegger and an anonymous referee for very helpful comments.

### References

- [1] L. Blum, F. Cucker, M. Shub, S. Smale, *Complexity and Real Computation*, Springer-Verlag, 1998.
- [2] R. Courant, D. Hilbert, *Methods of Mathematical Physics*, vol. II, Interscience, 1953, translated and revised from the German edition [1937].
- [3] E. Engeler, *Formal Languages: Automata and Structures*, Markham, 1968.
- [4] J. Hadamard, *Lectures on Cauchy’s Problem in Linear Partial Differential Equations*, Dover, 1952, translated from the French edition [1922].
- [5] I.N. Herstein, *Abstract Algebra*, 2nd edition, Macmillan, 1990.
- [6] S.C. Kleene, *Introduction to Metamathematics*, North Holland, 1952.
- [7] M. Pour-El, J.J. Richards, *Computability in Analysis and Physics*, Springer-Verlag, 1989.
- [8] H.L. Royden, *Real Analysis*, Macmillan, 1963.
- [9] W. Rudin, *Principle of Mathematical Analysis*, 3rd edition, McGraw-Hill, 1976.
- [10] V. Sperschneider, G. Antoniou, *Logic: A Foundation for Computer Science*, Addison-Wesley, 1991.
- [11] V. Stoltenberg-Hansen, J.V. Tucker, Computable rings and fields, in: E. Griffor (Ed.), *Handbook of Computability Theory*, Elsevier, 1999.
- [12] V. Stoltenberg-Hansen, J.V. Tucker, Concrete models of computation for topological algebras, *Theor. Comput. Sci.* 219 (1999) 347–378.
- [13] J.V. Tucker, J.I. Zucker, Computation by ‘while’ programs on topological partial algebras, *Theor. Comput. Sci.* 219 (1999) 379–420.
- [14] J.V. Tucker, J.I. Zucker, Computable functions and semicomputable sets on many-sorted algebras, in: *Handbook of Logic in Computer Science*, vol. 5, Oxford University Press, 2000, pp. 317–523.
- [15] J.V. Tucker, J.I. Zucker, Abstract versus concrete computation on metric partial algebras, *ACM Trans. Comput. Log.* 5 (2004) 611–668.
- [16] J.V. Tucker, J.I. Zucker, Computable total functions on metric algebras, universal algebraic specifications and dynamical systems, *J. Log. Algebr. Program.* 62 (2005) 71–108.
- [17] J.V. Tucker, J.I. Zucker, Continuity of operators on continuous and discrete time streams, *Theor. Comput. Sci.* 412 (2011) 3378–3403.
- [18] L. van den Dries, *Tame Topology and O-minimal Structures*, Cambridge University Press, 1998.
- [19] B.L. van der Waerden, *Modern Algebra*, vol. 1, 2nd edition, Frederick Ungar, 1964.
- [20] K. Weihrauch, *Computable Analysis, an Introduction*, Springer-Verlag, 2000.
- [21] Bo Xie, Characterizations of semicomputable sets of real numbers, M.Sc. Thesis, Department of Computing & Software, McMaster University, 2004, Technical Report CAS 04-06-JZ, McMaster University, August 2004.