Notions of s/comp in Topological Algebras over the Reals

Notions of Semicomputability in Topological Algebras over the Reals

By

Mark Armstrong, B.ASc.

A Thesis

Submitted to the School of Graduate Studies

in partial fulfilment of the requirements for the degree of

Master of Science

Department of Computing and Software

McMaster University

MASTER OF SCIENCE (2015) McMaster University

(Computing and Software) Hamilton, Ontario

TITLE: Notions of Semicomputability in Topological Algebras over the Reals

AUTHOR: Mark Armstrong, B.ASc.(McMaster University)

SUPERVISOR: Prof. Jeffery I. Zucker

NUMBER OF PAGES: viii, 60

Lay Abstract

We investigate to what extent certain well-known results of classical computability theory on the natural numbers hold in the context of generalised computability theories on the real numbers.

Abstract

Several results from classical computability theory (computability over discrete structures such as the natural numbers and strings over finite alphabets, due to Turing, Church, Kleene and others) have been shown to hold for a generalisation of computability theory over total abstract algebras, using for instance the model of ***While*** computation.

We present a number of results relating to computation on topological partial algebras, again using ***While*** computation. We consider several results from the classical theory in the context of topological algebra of the reals: closure of semicomputable sets under finite union (Chapter 4 Theorem 2, p.33), the equivalence of semicomputable and projectively (semi)computable sets (Chapter 5 Theorem 7, p.45), and Post's Theorem (i.e. a set is computable iff both it and its complement are semicomputable) (Appendix B Theorem 9, p.53).

This research has significance in the field of scientific computation, which is underpinned by computability on the real numbers. We will consider a "continuity principle", which states that computability should imply continuity; however, equality, order, and other total boolean-valued functions on the reals are clearly discontinuous. As we want these functions to be basic for the algebras under consideration, we resolve this incompatibility by redefining such functions to be partial, leading us to consider topological partial algebras.

Acknowledgements

I would like to earnestly thank my supervisor, Dr. Jeffery Zucker, for his tutelage throughout my Masters studies. His insights during my research and his guidance during writing made my thesis possible.

I also deeply thank Dr. Ming Quan Fu, whose Ph.D. thesis contained the basis for my Strucure Theorem. It was also of much guidance when putting together my introduction and background chapters. I came to know Ming Quan as Dr. Zucker's other computer science graduate student, and he also offered much advise and encouragement through my studies.

Many thanks to Professor Bröcker of Münster University for his input on the properties of semi-algebraic and basic sets.

Thanks to the members of my M.Sc. committee, Dr. Wolfram Kahl and Dr. Sanzheng Qiao for their careful and thorough review of the draft of my thesis and for the valuable comments they provided.

Thanks to my dear wife, Sarah, who together with our daughter Callie is the light of my life and the source of my joy. Thanks also to my parents, and to my brother and my sister-in-law, for all their support throughout my studies.

Contents

Chapter 1

Introduction

1.1 Generalising computability theory

In *classical computability theory*, many formalisms have been presented and been proven to be equivalent, including the formalism of Turing machines, λ- calculus, and the μ-recursive functions, presented by Alan Turing [Tur36], Alonzo Church [Chu36] and Stephen Kleene [Kle36] respectively during the 1930's. These all capture the informal notion of computation by a finite, deterministic algorithm on $\mathbb{N}$ or on Σ^* (the set of strings from a finite alphabet Σ).

We generalise the classical computability theory to other abstract structures, especially the domain of real numbers $\mathbb{R}$.

Our reason for generalising to abstract models is that scientific computation is done largely on the reals, so we wish to apply the techniques of classical computability theory to the set $\mathbb{R}$ of real numbers and similar sets.

An important difference between $\mathbb{R}$ and $\mathbb{N}$ is that real numbers can only be constructed as infinite objects, for instance, as infinite sequences of rational numbers. Thus when working with $\mathbb{R}$, at least with concrete computation mod-

els (described below), we must work with the ideas of finite approximations. Further the topology of the reals gives us the idea of "nearness", and the closeness of approximations. We will see that the topology of the reals is a crucial concept in computation over the reals.

A *model of computation* is a mathematical model of some general method of computing functions, or deciding membership of a set. We distinguish two main kinds of such models: abstract and concrete.

For abstract models of computation, the data are taken as primitives, so the programs and algorithms do not depend on representations.

Examples of abstract models of computation are high level programming language, flow charts and register machines over any algebra.

For concrete models data are given by representations, and so the programs and algorithms are highly dependent on the choice of representation. For example, the reals may have finite representations by (indices of) effective Cauchy sequences.

Examples of concrete models of computation are "tracking computability" [TZ04, TZ05], Grzegorczyk-Lacombe computability [Grz55, Grz57] and Weihrauch's Type 2 computability [Wei00].

An important part of our work in this thesis is to consider whether certain results from the classical theory still hold in the *generalised computability theory*. We also show the equivalence and inequivalence of certain abstract models of computation based on the ***While*** language over $\mathbb{R}$.

1.2 Overview of the thesis

In Chapter 2 we review many-sorted structures and algebras, relations and projections, topological partial algebras (in particular the algebra $\mathcal{R}$ on $\mathbb{R}$ with the ring structure of the reals), and the ***While***, $\boldsymbol{While}^{\mathsf{OR}}$ and $\boldsymbol{While}^{\exists\mathsf{N}}$ program-

ming languages over $\mathcal{R}$, as well as the projective and "starred" versions of those languages.

In Chapter 3 we give lemmas and definitions for the ***While*** language on $\mathcal{R}$ that we use to present a Structure Theorem for ***While***$(\mathcal{R})$ in Chapter 4. Using that Structure Theorem, we prove that the set of ***While***$(\mathcal{R})$ semicomputable sets is not closed under union.

In Chapter 5 we present results regarding the equivalence of models of computation on $\mathcal{R}$ based on the ***While*** language.

In Chapter 6 we present our conclusion, and some ideas for about future work.

In Appendix A we prove the equivalence of the starred versions of models based on the ***While*** language.

In Appendix B we show that another closure result from the classical theory, Post's Theorem, holds trivially in the case of $\mathcal{R}$, but does not hold more generally for partial algebras.

Chapter 2

Signatures; Algebras; the *While* language

We will study the computation of functions and relations by high level imperative programming languages based on the 'while' construct, applied to a many-sorted signature Σ. We give semantics for this language relative to a topological partial Σ-algebra A, and define the notions of *computability, semicomputability* and *projective semicomputability* for this language on A. Much of the material is taken from [TZ00, TZ15], adapted to partial algebras.

We begin by reviewing basic concepts: many-sorted signatures and algebras. Next we define the syntax and semantics of the ***While*** programming language. Then we present several extensions to this language.

2.1 Signatures

Definition 2.1.1 (Signature)**.** A many-sorted signature Σ is a pair $\langle \boldsymbol{Sort}(\Sigma), \boldsymbol{Func}(\Sigma) \rangle$ where

(a) $\boldsymbol{Sort}(\Sigma)$ is a finite set of basic types called *sorts* s, s',....

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

$$F\colon s_1 \times \cdots \times s_m \to s \qquad (m \geq 0)$$

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

Definition 2.1.2 (Product Type)**.** A *product type* of A has the form $s_1 \times \cdots \times s_m$, where $m \geq 0$ and $s_1, \ldots, s_m$ are sorts of A. We write $u, v, ...$ for product types. A *function type* has the form $u \to s$, where u is a product type.

Definition 2.1.3 (Standard Signature)**.** A signature is called *standard* if it includes the sorts and functions of the signature of the booleans:

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

All signatures used in this thesis are standard.

Definition 2.1.4 (ℕ-standard Signature)**.** A signature is called ℕ-*standard* if, in addition to being standard, it includes the sorts and functions of the signature of the naturals:

signature	$\Sigma(\mathcal{N})$
sorts	nat
functions	$0 : \rightarrow \mathsf{nat}$
	$\mathsf{suc} : \mathsf{nat} \rightarrow \mathsf{nat}$
	$\mathsf{eq}, \mathsf{less} : \mathsf{nat}^2 \rightarrow \mathbb{B}$

All signatures used in our main results are ℕ-standard[1].

2.2 Algebras

Definition 2.2.1 (Algebra)**.** For a signature Σ, a Σ-*algebra* A has, for each sort s of Σ, a non-empty set A_s, called the *carrier* of sort s, and for each function symbol $F : s_1 \times ... \times s_m \rightarrow s$, a function $F^A : A_{s_1} \times ... \times A_{s_1} \rightarrow A_s$.

Notation 2.2.2. We write $\Sigma(A)$ for the signature of an algebra A.

Notation 2.2.3. For a Σ-product type $u = s_1 \times ... \times s_m$, we write

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

Definition 2.2.4 (Total and Partial Algebras)**.** An algebra is *total* if f^A is total for all $f \in \boldsymbol{Func}(\Sigma)$; otherwise, it is *partial*.

[1]The signature used in Appendix B is not ℕ-standard, for the sake of the simplicity of the proof in that chapter.

The (total) algebras of the booleans and the naturals are as follows:

algebra	$\mathcal{B}$
carrier	$\mathbb{B}$
functions	$\mathsf{true}^{\mathsf{B}}, \mathsf{false}^{\mathsf{B}} : \ \to \mathbb{B}$
	$\mathsf{or}^{\mathsf{B}}, \mathsf{and}^{\mathsf{B}} : \mathbb{B}^2 \to \mathbb{B}$
	$\mathsf{cor}^{\mathsf{B}}, \mathsf{cand}^{\mathsf{B}} : \mathbb{B}^2 \to \mathbb{B}$
	$\mathsf{not}^{\mathsf{B}} : \mathbb{B} \to \mathbb{B}$

algebra	$\mathcal{N}$
carriers	$\mathbb{N}$
functions	$0^{\mathsf{N}} : \ \to \mathbb{N}$
	$\mathsf{suc}^{\mathsf{N}} : \mathbb{N} \to \mathbb{N}$
	$\mathsf{eq}^{\mathsf{N}}, \mathsf{less}^{\mathsf{N}} : \mathbb{N}^2 \to \mathbb{B}$

where the functions $\mathsf{true}^{\mathsf{B}}$, $\mathsf{false}^{\mathsf{B}}$, $\mathsf{and}^{\mathsf{B}}$, or^{B}, $\mathsf{not}^{\mathsf{B}}$, 0^{N}, S^{N}, eq^{N} and $\mathsf{less}^{\mathsf{N}}$ have their usual definitions, and the functions $\mathsf{cor}^{\mathsf{B}}$ and $\mathsf{cand}^{\mathsf{B}}$ are defined as in Definition 2.2.5.

We will often write $\vee$ and $\wedge$ in place of or and and, $\overset{c}{\vee}$ and $\overset{c}{\wedge}$ in place of cor and cand and $\neg$ in place of not, and drop the superscripts $\cdot^{\mathsf{B}}$ and $\cdot^{\mathsf{N}}$ where unambiguous.

Definition 2.2.5. The semantics of the "conditional operators"[2] $\overset{c}{\vee}$ and $\overset{c}{\wedge}$ are as follows:

$[\![b_1 \overset{c}{\vee} b_2]\!]^{\mathcal{B}}\sigma$

$[\![b_1]\!]^{\mathcal{B}}\sigma$ \ $[\![b_2]\!]^{\mathcal{B}}\sigma$	$\mathsf{true}^{\mathsf{B}}$	$\mathsf{false}^{\mathsf{B}}$	$\uparrow$
$\mathsf{true}^{\mathsf{B}}$	$\mathsf{true}^{\mathsf{B}}$	$\mathsf{true}^{\mathsf{B}}$	$\mathsf{true}^{\mathsf{B}}$
$\mathsf{false}^{\mathsf{B}}$	$\mathsf{true}^{\mathsf{B}}$	$\mathsf{false}^{\mathsf{B}}$	$\uparrow$
$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$

$[\![b_1 \overset{c}{\wedge} b_2]\!]^{\mathcal{B}}\sigma$

$[\![b_1]\!]^{\mathcal{B}}\sigma$ \ $[\![b_2]\!]^{\mathcal{B}}\sigma$	$\mathsf{true}^{\mathsf{B}}$	$\mathsf{false}^{\mathsf{B}}$	$\uparrow$
$\mathsf{true}^{\mathsf{B}}$	$\mathsf{true}^{\mathsf{B}}$	$\mathsf{false}^{\mathsf{B}}$	$\uparrow$
$\mathsf{false}^{\mathsf{B}}$	$\mathsf{false}^{\mathsf{B}}$	$\mathsf{false}^{\mathsf{B}}$	$\mathsf{false}^{\mathsf{B}}$
$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$

i.e., the operators $\overset{c}{\vee}$ and $\overset{c}{\wedge}$ are "evaluated from the left".

The definition of $[\![\cdot]\!]^{\mathcal{B}}\sigma$ is given in Section 2.8.

Definition 2.2.6 (Standard Algebra)**.** An algebra is called *standard* if it is an expansion of $\mathcal{B}$ and any equality operators, for values on which they are defined, they are the identity of their respective sorts.

All algebras in this thesis are standard.

Definition 2.2.7 ($\mathbb{N}$-standard Algebra)**.** An algebra is called $\mathbb{N}$-*standard* if it is an expansion of $\mathcal{N}$.

All algebras used in our main results are $\mathbb{N}$-standard[3].

[2]The operators $\overset{c}{\vee}$ and $\overset{c}{\wedge}$ are called conditional operators because they can be simulated using conditional statements. Thus, in the ***While*** language, $b_1 \overset{c}{\vee} b_2$ can be simulated by "$\mathsf{if}\ b_1\ \mathsf{then\ true\ else}\ b_2\ \mathsf{fi}$" and $b_1 \overset{c}{\wedge} b_2$ by "$\mathsf{if}\ \neg b_1\ \mathsf{then\ false\ else}\ b_2\ \mathsf{fi}$".

[3]See footnote 1.

2.3 Relations and projections

Let Σ be any signature and A any Σ-algebra.

Definition 2.3.1 (Relation)**.** A *relation* on A of type u is a subset of A^u. We write $R : u$ if R is a relation of type u.

Definition 2.3.2 (Complement of a Relation)**.** The *complement of* R in A is the relation

$$R^c = A^u \backslash R = \{a \in A^u \mid a \notin R\}$$

Definition 2.3.3 (Projection)**.** Let R be a relation of type $u = s_1 \times ... \times s_m$ where $m > 0$. Let $\vec{i} = i_1, ..., i_r$ be a list of numbers such that $1 \leq i_1 < ... < i_r \leq m$, and let $\vec{j} = j_1, ..., j_{m-r}$ be the list $\{1, ..., m\}\backslash\vec{i}$. Then the projection of R off of i is the relation $S : s_{j_1} \times ... \times s_{j_{m-r}}$ such that:

$$S(x_{j_1}, ..., x_{j_{m-r}}) \iff \exists x_{i_1}, ..., x_{i_r} : s_{i_1}, ..., s_{i_r}, R(x_1, ..., x_m).$$

2.4 Topological partial algebras

Recall the definition of continuity of partial functions[4]:

Definition 2.4.1 (Continuity)**.** Given two topological spaces X and Y, a partial function $f : X \rightharpoonup Y$ is *continuous* if for every open $V \subseteq Y$,

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

is open in X.

Definition 2.4.2 (Topological partial algebra)**.** A *topological partial algebra* is a partial Σ-algebra with topologies on the carriers such that each of the basic Σ-functions is continuous. The carriers $\mathbb{B}$ and $\mathbb{N}$, if present, have the discrete topology.

[4]For more information on topologies, topological spaces and continuity, refer to any standard topology text, such as Rudin's *Principles of Mathematical Analysis*.

Remark 2.4.3 (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 function on A [TZ99, TZ00].

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

$$\textbf{\textit{computability}} \implies \textbf{\textit{continuity}}.$$

This principal is discussed in [TZ99, TZ04].

2.5 The algebra $\mathcal{R}$ of reals

In the following sections, we work mostly with the following topological algebra[5]:

algebra	$\mathcal{R}$
carriers	$\mathbb{R}, \mathbb{B}, \mathbb{N}$
functions	$0^{\mathsf{R}}, 1^{\mathsf{R}} : \ \to \mathbb{R}$
	$\mathsf{plus}^{\mathsf{R}}, \mathsf{times}^{\mathsf{R}} : \mathbb{R}^2 \to \mathbb{R}$
	$0^{\mathsf{N}} : \ \to \mathbb{N}$
	$\mathsf{suc}^{\mathsf{N}} : \mathbb{N} \to \mathbb{N}$
	$\mathsf{true}, \mathsf{false} : \ \to \mathbb{B}$
	$\mathsf{or}, \mathsf{and} : \mathbb{B}^2 \to \mathbb{B}$
	$\mathsf{cor}, \mathsf{cand} : \mathbb{B}^2 \to \mathbb{B}$
	$\mathsf{not} : \mathbb{B} \to \mathbb{B}$
	$\mathsf{eq}^{\mathsf{N}}, \mathsf{less}^{\mathsf{N}} : \mathbb{N}^2 \to \mathbb{B}$
	$\mathsf{eq}^{\mathsf{R}}, \mathsf{less}^{\mathsf{R}} : \mathbb{R}^2 \to \mathbb{B}$

[5] In [Fu14], this algebra was called $\mathcal{R}_0$; $\mathcal{R}$ was the algebra which also included the inverse operation for the reals.

where the functions 0^{R}, 1^{R}, $\mathsf{plus}^{\mathsf{R}}$, $\mathsf{times}^{\mathsf{R}}$, 0^{N}, S^{N}, $\mathsf{true}^{\mathsf{B}}$, $\mathsf{false}^{\mathsf{B}}$, $\mathsf{and}^{\mathsf{B}}$, or^{B}, $\mathsf{not}^{\mathsf{B}}$, eq^{N} and $\mathsf{less}^{\mathsf{N}}$ have their usual definitions, the functions cor and cand are defined as in Definition 2.2.5, and the function eq^{R} and $\mathsf{less}^{\mathsf{R}}$ are defined as in Remark 2.5.1.

We will often write $=$ and $<$ in place of eq^{R} and $\mathsf{less}^{\mathsf{R}}$, and drop the superscripts $\cdot^{\mathsf{R}}$, $\cdot^{\mathsf{B}}$ and $\cdot^{\mathsf{N}}$ where unambiguous

The signature $\Sigma(\mathcal{R})$ can be inferred from the above, with real as the sort of $\mathbb{R}$.

Remark 2.5.1. $\mathcal{R}$ is a partial algebra, with the basic partial functions eq^{R} and $\mathsf{less}^{\mathsf{R}}$, where for $x, y \in \mathbb{R}$:

$$[\![\mathsf{eq}^{\mathsf{R}}(x,y)]\!]^{\mathcal{R}}\sigma \simeq \begin{cases} \uparrow & \text{if } [\![x]\!]^{\mathcal{R}}\sigma = [\![y]\!]^{\mathcal{R}}\sigma \\ \mathsf{false} & \text{o/w} \end{cases}$$

and

$$[\![\mathsf{less}^{\mathsf{R}}(x,y)]\!]^{\mathcal{R}}\sigma \simeq \begin{cases} \mathsf{true} & \text{if } [\![x]\!]^{\mathcal{R}}\sigma < [\![y]\!]^{\mathcal{R}}\sigma \\ \mathsf{false} & \text{if } [\![x]\!]^{\mathcal{R}}\sigma > [\![y]\!]^{\mathcal{R}}\sigma \\ \uparrow & \text{o/w.} \end{cases}$$

The definition of $[\![\cdot]\!]^{\mathcal{R}}\sigma$ is given in section 2.8.

By contrast, the basic functions eq^{N} and $\mathsf{less}^{\mathsf{N}}$ on $\mathbb{N}$, are total.

Notation 2.5.2. The symbol '$\simeq$' denotes Kleene equality, where the two sides are either both defined and equal, or both undefined.

Discussion 2.5.3 (Motivation for use of partial functions)**.** We present our motivation for using partial functions in two ways. The first is a discussion of continuity of comparison operators on $\mathbb{R}$. The second is a *Gedankenexperiment* involving concrete models of computation.

(1) The total versions of the comparison operators eq^{R} and $\mathsf{less}^{\mathsf{R}}$ on $\mathbb{R}$ are not continuous. (By contrast any comparison operators on $\mathbb{N}$ are continuous, because $\mathbb{N}$ has the discrete topology). Continuity of these comparison operators is important due to the Continuity Principle (Remark 2.4.3) and our definition of topological partial algebras (Definition 2.4.2), which requires all basic operators to be continuous.

(2) Consider now the task of determining whether for two real variables $\mathtt{x}$ and $\mathtt{y}$, $\mathtt{x} = \mathtt{y}$, in some concrete model of computation. For this example, let us use as a representation of these real numbers effective Cauchy sequences of rationals $(r_0, r_1, r_2, ...)$ and $(s_0, s_1, s_2, ...)$. We assume for our convenience that the sequences are "fast", i.e.,

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

and similarly for (s_n). Now suppose that for $n = 1, 2, 3, ...$ the inputs r_n and s_n are observed (from some device) at n time units. Now $\mathtt{x} < \mathtt{y}$ is true at σ iff for some n, $r_n + 2 \cdot 2^{-n} < s_n$, and this can be determined in a finite amount of time. Correspondingly, $\mathtt{x} = \mathtt{y}$ is true iff for all n, $|r_n - s_n| \ < 2 \cdot 2^{-n}$, but this cannot be determined in a finite amount of time. So from this example it is natural for comparison operators on $\mathtt{x}$ and $\mathtt{y}$ to diverge in cases when $\mathtt{x} = \mathtt{y}$.

Remark 2.5.4. Throughout the main results of our thesis we focus on functions on $\mathbb{R}^2$. There are many problems to consider on $\mathbb{R}^2$ that are not present for functions on $\mathbb{R}$; for instance, ***While*** semicomputable sets on $\mathbb{R}$ are closed under union[6].

[6]This is clear from the characterization of sets on $\mathbb{R}$ given in [XFZ15].

2.6 The algebra $\mathcal{R}^*$

The algebra $\mathcal{R}^*$ is formed from $\mathcal{R}$ by adding the carriers $\mathbb{R}^*$, $\mathbb{N}^*$ and $\mathbb{B}^*$ (of sorts real^*, nat^* and bool^* respectively) consisting of all *finite sequences* or *arrays* of reals, naturals and booleans, together with certain standard constants and operations for the empty array, updating of arrays, etc.

The significance of arrays for computation is that they provide finite but unbounded memory. The reason for introducing the starred sort real^* is the lack of effective coding of finite sequences from $\mathbb{R}$ (unlike the case of $\mathbb{N}$).

We make use of $\mathcal{R}^*$ to simplify one of our results. However, despite the convenience of the starred sort real^*, the use of $\mathcal{R}^*$ is not essential as it is not strictly stronger for computation than $\mathcal{R}$ (we outline a proof of this fact in Appendix A). As such, we omit the precise definition of $\mathcal{R}^*$, which can be found in [TZ00, TZ15].

2.7 The *While* programming language

As has been mentioned, we will study the computation of functions and relations by high level imperative programming languages based on the 'while' construct.

For the remainder of this section, let A be a standard algebra with signature Σ.

We begin with the syntax of Σ-terms.

Notation 2.7.1. We use '$\equiv$' to denote syntactic identity between two expressions.

Definition 2.7.2 (Σ-variables)**.** For each Σ-sort s, there are *variables* $\mathtt{x}^s, \mathtt{y}^s, \ldots$ of sort s. $\boldsymbol{Var}_s(\Sigma)$ is the set of variables of sort s, and $\boldsymbol{Var}(\Sigma)$ is the set of all Σ-variables.

Definition 2.7.3 (Σ-terms)**.** $\boldsymbol{Tm}(\Sigma)$ is the set of *Σ-terms*: $t, ...$ and $\boldsymbol{Tm}_s(\Sigma)$ is the set of Σ-terms of sort s: $t^s, ...$. We define this using modified BNF:

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

where F is a Σ-function symbol of type $s_1 \times ... \times s_m \to s$ ($m \geq 0$).

Now we consider statements and procedures for the ***While*** language.

Definition 2.7.4 (Statements)**.** $\boldsymbol{Stmt}(\Sigma)$ is the class of *Σ-statements* $S, ...$ generated by:

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

where $\mathtt{x} := t$ denotes simultaneous assignment, i.e., for some $m > 0$,
$\mathtt{x} \equiv (\mathtt{x}_1, ..., \mathtt{x}_m)$ and $t \equiv (t_1, ..., t_m)$ are variable and term tuples of the same product type, with the condition that $\mathtt{x}_i \not\equiv \mathtt{x}_j$ for $i \neq j$; and b is a boolean term.

Definition 2.7.5 (Procedures)**.** $\boldsymbol{Proc}(\Sigma)$ is the class of *Σ-procedures* $P, ...$ of the form:

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

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

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

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

(i) $\mathtt{a}$, $\mathtt{b}$ and $\mathtt{c}$ each consist of distinct variables, and they are pairwise disjoint,

(ii) every variable occurring in the body S must be declared in D (among $\mathtt{a}$, $\mathtt{b}$ or $\mathtt{c}$).

If $\mathtt{a} : u$ and $\mathtt{b} : v$, then P has type $u \to v$, written $P : u \to v$.

For brevity, we will write ***Tm*** for $\boldsymbol{Tm}(\Sigma)$, ***While*** for $\boldsymbol{While}(\Sigma)$, etc.

2.8 States

Definition 2.8.1 (State)**.** A *state* is a family $\langle \sigma_s \mid s \in \boldsymbol{Sort}(\Sigma) \rangle$ of functions $\sigma_s : \boldsymbol{Var}_s \to A_s$. $\boldsymbol{State}(A)$ is the set of states on A, with elements $\sigma, \ldots$.

Notation 2.8.2. We will write $\sigma(\mathtt{x})$ for $\sigma_s(\mathtt{x})$ when $\mathtt{x} \in \boldsymbol{Var}_s$. We also write, for tuples $\mathtt{x} \equiv (\mathtt{x}_1, ..., \mathtt{x}_m)$, $\sigma[\mathtt{x}]$ in place of $(\sigma(\mathtt{x}_1), ..., \sigma(\mathtt{x}_m))$.

Definition 2.8.3 (Variant of a state)**.** Let σ be a state over A, and for some Σ-product type u, let $\mathtt{x} \equiv (\mathtt{x}_1, ..., \mathtt{x}_n)$ and $a = (a_1, ..., a_n) \in A^u$ (for $n \geq 1$). We define $\sigma\{\mathtt{x}/a\}$ to be the state over A formed from σ by replacing its value at $\mathtt{x}_i$ by a_i for $i = 1, ..., n$. That is, for all variables $\mathtt{y}$:

$$\sigma\{\mathtt{x}/a\}(\mathtt{y}) = \begin{cases} \sigma(\mathtt{y}) & \text{if } \mathtt{y} \not\equiv \mathtt{x}_i \text{ for } i = 1, ..., n \\ a_i & \text{if } \mathtt{y} \equiv \mathtt{x}_i \end{cases}$$

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

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

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

Definition 2.8.4 (Semantics of terms)**.** The definition of $[\![t]\!]^A\sigma$ is by structural induction on Σ-terms t:

- $[\![\mathtt{x}]\!]^A\sigma = \sigma(\mathtt{x})$
- $[\![F(t_1, ..., t_m)]\!]^A\sigma \simeq \begin{cases} F([\![t_1]\!]^A\sigma, ..., [\![t_m]\!]^A\sigma) & \text{if } [\![t_i]\!]^A\sigma \downarrow \text{ for all } i = 1, ..., m \\ \uparrow & \text{o/w} \end{cases}$

Notation 2.8.5. We write $[\![t]\!]^A\sigma \uparrow$ if it diverges and $[\![t]\!]^A\sigma \downarrow$ if it converges, which we can think of as the computation of $[\![t]\!]^A\sigma$ not halting and halting respectively. We also write $[\![t]\!]^A\sigma \downarrow a$ to mean that the evaluation of $[\![t]\!]^A\sigma$ converges to a value a.

As with Kleene equality at the meta-level (Notation 2.5.2), we make use of Kleene equality on the meaning of terms.

Notation 2.8.6. $[\![t_1]\!]^A\sigma \simeq [\![t_2]\!]^A\sigma$ means that either $[\![t_1]\!]^A\sigma$ and $[\![t_1]\!]^A\sigma$ both converge to the same value, or both diverge (cf. [Kle52, §63]).

2.9 Semantics of the *While* language

In this section we define functions which give the meaning of ***While*** statements S and procedures P, and some of the functions used to define those functions. The definitions of these functions is standard [TZ00, §§3.4-3.6, §§3.14] and lengthy, and so we only give descriptions, not definitions.

We will make use of the *computation step* function:

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

$\boldsymbol{Comp}^A(S, \sigma, n)$ is the nth step, or the state at the nth time cycle, in the computation of S on A, starting at state σ. The symbol '$*$' indicates the computation is over.

From the computation step function we can easily define another function we will use, the *computation length* function:

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

as follows:

$$\boldsymbol{CompLen}^A(S, \sigma) = \begin{cases} \text{least } n \text{ such that } \boldsymbol{Comp}^A(S, \sigma, n) = * & \text{if such an } n \text{ exists} \\ \infty & \text{o/w} \end{cases}.$$

We will also make use of the *statement remainder* function:

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

$\boldsymbol{Rem}^A(S, \sigma, n)$ is the statement about to be executed at step n of the computation of S on A, starting at state σ (or skip when the computation is over); this statement is called the "remainder" of S.

Now we may define the ideas of *snapshots*. We define the snapshot function:

$$\boldsymbol{Snap}^A(S, \sigma, n) : \boldsymbol{Stmt} \times \boldsymbol{State}(A) \times \mathbb{N} \rightharpoonup ((\boldsymbol{State}(A) \cup \{*\}) \times \boldsymbol{Stmt})$$

as

$$\boldsymbol{Snap}^A(S, \sigma, n) = (\sigma^n, S^n)$$

where

$$\sigma^n = \boldsymbol{Comp}^A(S, \sigma, n) \text{ and } S^n = \boldsymbol{Rem}^A(S, \sigma, n).$$

$\boldsymbol{Snap}^A(S, \sigma, n)$ is the snapshot of the computation of S at stage n, i.e. the pair (σ^n, S^n) where σ^n is the nth step in the computation of S and S^n is the statement about to be executed at state n.

From the snapshot function we may define the *snapshot sequence* generated by S at σ:

$$(\boldsymbol{\sigma}, \boldsymbol{S}) = ((\sigma^0, S^0), (\sigma^1, S^1), (\sigma^2, S^2), ...)$$

$(\boldsymbol{\sigma}, \boldsymbol{S})$ is an infinite sequence of snapshots generated by S at σ. If S halts on σ, then eventually the sequence repeats (σ', skip).

The meaning of a ***While*** statement S, written $[\![S]\!]^A$, is a partial state transformation on an algebra A:

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

Let $l = \boldsymbol{CompLen}^A(S, \sigma)$; then:

$$[\![S]\!]^A(\sigma) \simeq \begin{cases} \boldsymbol{Comp}^A(S, \sigma, l) & \text{if } l \neq \infty \\ \uparrow & \text{o/w} \end{cases}.$$

The meaning of a ***While*** procedure

$$P \equiv \mathsf{proc\ in\ a} : u\ \mathsf{out\ b} : v\ \mathsf{aux\ c} : w\ \mathsf{begin}\ S\ \mathsf{end}\ u \to v$$

is written $[\![P]\!]^A : A^u \to A^v$, and defined as follows. For $a \in A^u$, let σ be any state on A such that $\sigma[\mathsf{a}] = a$. Then

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

2.10 *While* computability and semicomputability

Definition 2.10.1 (***While*** computable function)**.** Let A be a standard algebra.

(a) A function $f : A^u \rightharpoonup A_s$ is said to be *computable* (on A) by a ***While*** procedure $P : u \to s$ if $f = P^A$.

(b) ***While***(A) is the class of functions ***While*** computable on A.

Definition 2.10.2 (Halting set)**.** The *halting set* of a procedure $P : u \to v$ on A is the set

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

Definition 2.10.3 (***While*** semicomputable set)**.** A set $R \subseteq A^u$ is ***While*** *semicomputable* on A if it is the halting set on A for some ***While*** procedure.

2.11 Extending *While* to *While*$^{\mathsf{OR}}$ and *While*$^{\exists\mathsf{N}}$

In preparation for the theorems in Chapter 4 and Chapter 5, we give the semantics of strong disjunction and infinite disjunctions to introduce the extensions ***While***$^{\mathsf{OR}}$ and ***While***$^{\exists\mathsf{N}}$ extensions of the ***While*** language.

The motivation of these extensions is that our model of ***While*** computation on $\mathcal{R}$, the partial operations leave us unable to implement *interleaving* or *dovetailing*. The problem is that when interleaving two processes, one may converge and the other diverge locally (because of the partial operations). The resulting process will then diverge, whereas we would want it to converge. Thus, as we will see in one of the results of Chapter 4, the union of two semicomputable sets is not necessarily semicomputable! In concrete models do not have this deficiency. The extensions $\textbf{\textit{While}}^{\mathsf{OR}}$ and $\textbf{\textit{While}}^{\exists\mathsf{N}}$ correct this deficiency.

The $\textbf{\textit{While}}^{\mathsf{OR}}$ language is created from ***While*** by introducing the strong (Kleene) disjunction operator '∇', where $b_1 \nabla b_2$ converges to true if either b_1 or b_2 converge to true, even if the other diverges.

The $\textbf{\textit{While}}^{\exists\mathsf{N}}$ language is created from ***While*** by introducing a strong existential quantification construct over the naturals:

$$\mathtt{x}^b := \ \exists \mathtt{z}\ P(t, \mathtt{z})$$

where $z : \mathsf{nat}$ and P is a boolean valued procedure. Its semantics are as follows:

$$[\![\exists \mathtt{z}\ P(t, \mathtt{z})]\!]^A\sigma \simeq \begin{cases} \mathsf{true} & \text{if } P([\![t]\!]^A\sigma, n) \downarrow \mathsf{true} \text{ for some } n \\ \uparrow & \text{o/w.} \end{cases}$$

We also include the strong disjunction operator in the $\textbf{\textit{While}}^{\exists\mathsf{N}}$ language.

By means of these constructs, interleaving of processes may be simulated. The $\textbf{\textit{While}}^{\mathsf{OR}}$ language allows for interleaving of an arbitrary but finite number of processes, and the $\textbf{\textit{While}}^{\exists\mathsf{N}}$ language allows for interleaving of *infinitely* many processes.

2.12 Extending *While*, *While*$^{\mathsf{OR}}$ and *While*$^{\exists\mathsf{N}}$ to their starred versions

Recall the algebra $\mathcal{R}^*$ (section 2.6). We now construct the $\boldsymbol{While}^*(\mathcal{R})$, $\boldsymbol{While}^{\mathsf{OR}*}(\mathcal{R})$ and $\boldsymbol{While}^{\exists\mathsf{N}*}(\mathcal{R})$.

Definition 2.12.1 (Simple and starred variables)**.** We call the variables of sort real, nat and bool *simple*, and the variables of sort real^*, nat^* and bool^* *starred*.

Definition 2.12.2 (The $\boldsymbol{While}^*(\mathcal{R})$ language)**.** A $\boldsymbol{While}^*(\mathcal{R})$, $\boldsymbol{While}^{\mathsf{OR}*}(\mathcal{R})$ or $\boldsymbol{While}^{\exists\mathsf{N}*}(\mathcal{R})$ procedure is respectively a $\boldsymbol{While}(\mathcal{R}^*)$, $\boldsymbol{While}^{\mathsf{OR}}(\mathcal{R}^*)$ or $\boldsymbol{While}^{\exists\mathsf{N}}(\mathcal{R}^*)$ procedure for which the *input* and *output* variables are simple. However, the auxiliary variables may be starred.

2.13 Encoding of syntax

We assume given a family of effective numerical codings for each of the classes of syntactic expressions over Σ. We write $\ulcorner E \urcorner$ for the code of an expression E. We make the following assumptions about the coding:

- $\ulcorner E \urcorner$ increases strictly with the complexity of E, and so (e.g.), the code of an expression is larger than those of its subexpressions.
- Sets of codes of the various syntactic classes, and of their respective subclasses, such as $\{\ulcorner t \urcorner \mid t \in \boldsymbol{Term}\}$, $\{\ulcorner t \urcorner \mid t \in \boldsymbol{Term}_s\}$, $\{\ulcorner S \urcorner \mid S \in \boldsymbol{Stmt}\}$, $\{\ulcorner S \urcorner \mid S \text{ is an assignment}\}$, etc., are primitive recursive.
- We can go primitive recursively from codes of expressions to codes of their immediate subexpressions and vice versa; thus, for example, $\ulcorner S_1 \urcorner$ and $\ulcorner S_2 \urcorner$ are primitive recursive in $\ulcorner S_1; S_2 \urcorner$, and conversely.

In short, we can primitive recursively simulate all operations involved in processing the syntax of the programming language.

Chapter 3

Semantic disjointedness; Engeler's Lemma

3.1 Engeler's Lemma

The following lemma is of vital importance to proving our Structure Theorem for ***While***$(\mathcal{R})$ semicomputable sets [Eng68], [TZ00].

Lemma 3.1.1 (Engeler's Lemma for ***While***)**.** *If a relation $R \subseteq A^u$ is **While** semicomputable over a standard partial Σ-algebra A, then R can be expressed as the disjunction of an effective countable sequence of Σ-booleans*[1] *over A.*

i.e.,

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

for an effective sequence for booleans $(b_1, b_2, b_3, ...)$.

We also need the following concept and lemma.

[1] That is, Σ-terms of sort bool.

Definition 3.1.2 (Semantic Disjointedness)**.** A sequence $(b_0, b_1, b_2, ...)$ of boolean terms is *semantically disjoint* over A if for any state σ over A and any n,

$$[\![b_n]\!]^A\sigma \downarrow \mathsf{true} \implies \forall i \neq n, [\![b_i]\!]^A\sigma \downarrow \mathsf{false}.$$

The following lemma was proved in [XFZ15, §4].

Lemma 3.1.3 (Semantic Disjointedness Lemma)**.** *The sequence of computable boolean terms generated from a* ***While*** *computation tree S by the construction using computation trees in the proof of Engeler's Lemma*[2] *is semantically disjoint.*

3.2 Canonical form for booleans over $\mathcal{R}$

In our proof of our Structure Theorem for ***While***$(\mathcal{R})$ semicomputable sets, we require a canonical form for booleans over $\mathcal{R}$.

Lemma 3.2.1 (Canonical form for booleans over $\mathcal{R}$)**.** *An $\mathcal{R}$-boolean with variables of sort* real *only is effectively semantically equivalent to a boolean combination of equations and inequalities of the form:*

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

where p and q are polynomials in $\boldsymbol{x}$ of degree > 0.

The proof of Lemma 3.2.1 resembles that of a similar lemma for booleans over $\mathcal{R}^{\mathsf{OR}}$ [Fu14, §§4.1].

3.3 Semi-algebraic and basic sets

We introduce the concepts of semi-algebraic and basic sets, which are fundamental to our results. We consider these sets on $\mathbb{R}^2$, though they can clearly be generalised to $\mathbb{R}^n$ for any $n \geq 0$.

[2] [XFZ15, Lemma 4.3.1]

Definition 3.3.1 (Semi-algebraic set)**.** A *semi-algebraic set* is a *finite union* of sets of the form

$$\{x \in \mathbb{R}^2 \mid p_1(x) > 0, ..., p_k(x) > 0, q_1(x) = 0, ..., q_l(x) = 0\} \ (k, l \geq 0)$$

where $p_1, ..., p_k, q_1, ..., q_l$ are polynomials with integer coefficients.

Definition 3.3.2 (Basic set)**.** A *basic set*[3] is a particular kind of semi-algebraic set, of the form

$$\{x \in \mathbb{R}^2 \mid p_1(x) > 0, ..., p_k(x) > 0\} \ (k \geq 0)$$

where $p_1, ..., p_k$ are polynomials with integer coefficients.

Remark 3.3.3. All basic sets are open.

Remark 3.3.4. Basic (open) sets are closed under intersection.

Lemma 3.3.5. *Given a polynomial $p(\mathtt{x})$ on $\mathbb{R}^2$, there are disjoint basic sets B^+, B^- and a semi-algebraic set D, such that on B^+, $p > 0$, on B^-, $p < 0$, and on D, $p = 0$, and $B^+ \cup B^- \cup D = \mathbb{R}^2$*

Proof. Clear. □

3.4 Positive and negative sets

Notation 3.4.1. For a pair of variables $\mathtt{x} \equiv (\mathtt{x}_1, \mathtt{x}_2) : \mathsf{real}^2$, let $\boldsymbol{Bool}(\mathtt{x})$ be the set of $\Sigma(\mathcal{R})$-booleans with no free variables other than $\mathtt{x}$.

As mentioned in Remark 2.5.4, we focus entirely on functions over $\mathbb{R}^2$ throughout this section.

We use the notions of *positive*, *negative* and *divergent* sets of booleans:

[3] In some texts, basic sets as we define them are called basic *open* sets; we work only with basic open sets, and so often omit the "open".

Definition 3.4.2. For any $b \in \textbf{\textit{Bool}}(\mathtt{x})$, let:

$$PS(b) =_{df} \{x \in \mathbb{R}^2 \mid b[x] = \mathtt{true}\}$$

$$NS(b) =_{df} \{x \in \mathbb{R}^2 \mid b[x] = \mathtt{false}\}$$

$$DS(b) =_{df} \{x \in \mathbb{R}^2 \mid b[x] \uparrow\}.$$

We call $PS(b)$, $NS(b)$ and $DS(b)$ the positive, negative and divergent sets of b respectively.

Chapter 4

$\boldsymbol{While}(\mathcal{R})$ semicomputable sets: Structure Theorem and failure of closure under union

In this chapter we begin by extending the Partition Lemma for $\boldsymbol{While}(\mathcal{R})$ semicomputability given in [Fu14], which we then use to give a Structure Theorem for $\boldsymbol{While}(\mathcal{R})$ semicomputability. Then, using that Structure Theorem, we give an example of two $\boldsymbol{While}(\mathcal{R})$ semicomputable sets whose union is not $\boldsymbol{While}(\mathcal{R})$, semicomputable, thus disproving the closure of $\boldsymbol{While}(\mathcal{R})$ semicomputable sets under union.

4.1 Partition Lemma for $\boldsymbol{While}(\mathcal{R})$

In addition to the lemmas given in chapter 3, we require one additional lemma in order to prove our Structure Theorem for $\boldsymbol{While}(\mathcal{R})$ semicomputable sets. This Partition Lemma is a strengthening of a similar lemma given in [Fu14, §§4.3], made possible by restricting our attention to ***While***.

Lemma 4.1.1 (Partition Lemma for booleans on $\mathcal{R}$)**.** *Consider any boolean* $b \in \boldsymbol{Bool}(\mathtt{x})$.[1] *We may construct positive and negative sets*[2] *for* b *expressed as:*

$$PS(b) = \bigcup_{i=1}^{k} B_i^+$$
$$NS(b) = \bigcup_{j=1}^{l} B_j^-.$$

where B_i^+, B_j^- *are basic (open) sets, and*

$$\boxed{\begin{array}{l} B_i^+ \cap B_j^- = \emptyset \text{ for } i = 1, ..., k \text{ and } j = 1, ..., l \\ B_{i_1}^+ \cap B_{i_2}^+ = \emptyset \text{ for } i_1 \neq i_2 \\ B_{j_1}^- \cap B_{j_2}^- = \emptyset \text{ for } j_1 \neq j_2 \end{array}}.$$

Remark 4.1.2. In [XFZ15] and [Fu14], basic sets for b were constructed such that only the first of the intersection properties hold, but for our purposes, considering the specific case of the ***While*** language, it is important to explicitely have all three.

[1] Notation 3.4.1
[2] For our purpose, the form of the divergent set of b, $DS(b)$, is unimportant.

Discussion 4.1.3. Before giving the proof, we first consider some examples of positive and negative sets in $\mathbb{R}^2$.

Consider the polynomials $p_1 = -x_1^2 - x_2^2 + 1$ and $p_2 = -(x_1 - 1)^2 - x_2^2 + 1$, and the booleans $b_1 \equiv p_1(x_1, x_2) > 0$ and $b_2 \equiv p_2(x_1, x_2) > 0$. Then:

$$B_1 = PS(b_1) = \{(x_1, x_2) \mid p_1(x_1, x_2) > 0\}$$
$$B_2 = PS(b_2) = \{(x_1, x_2) \mid p_2(x_1, x_2) > 0\}$$

Figure 4.1: $B_1 = PS(b_1)$

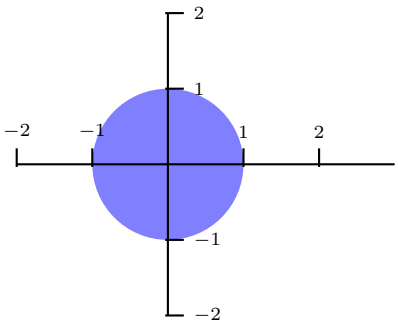
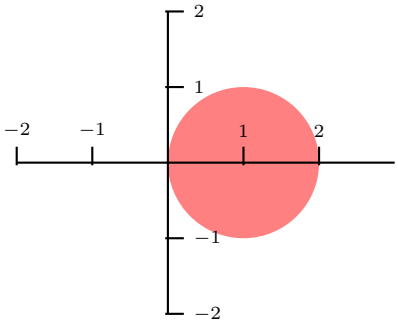


Figure 4.2: $B_2 = PS(b_2)$



B_1 and B_2 are basic sets, and can be easily be seen to be ***While***$(\mathcal{R})$ semi-computable. They are pictured in Figures 4.1 and 4.2 respectively.

Figure 4.3: $PS(b_1 \overset{c}{\vee} b_2)$

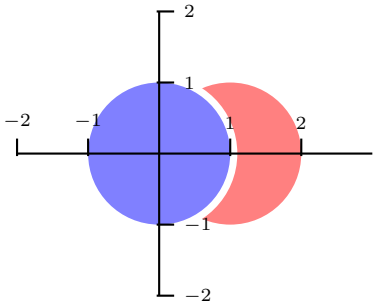
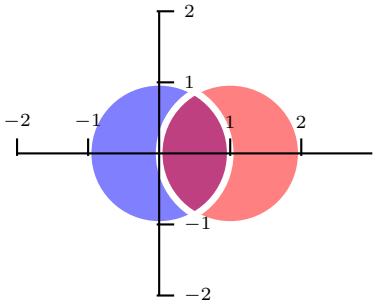


Figure 4.4: $PS(b_1 \vee b_2)$



The sets $PS(b_1 \overset{c}{\vee} b_2)$ and $PS(b_1 \vee b_2)$, pictured in Figures 4.3 and 4.4 respectively, are easily seen to be semicomputable. These sets can be represented as the union of two and three disjoint basic sets respectively[3]:

$$\begin{aligned} PS(b_1 \overset{c}{\vee} b_2) = &\{(x_1, x_2) \mid p_1(x_1, x_2) > 0\} \\ &\cup \{(x_1, x_2) \mid p_2(x_1, x_2) > 0, p_1(x_1, x_2) < 0\} \\ PS(b_1 \vee b_2) = &\{(x_1, x_2) \mid p_1(x_1, x_2) > 0, p_2(x_1, x_2) < 0\} \\ &\cup \{(x_1, x_2) \mid p_2(x_1, x_2) > 0, p_1(x_1, x_2) < 0\} \\ &\cup \{(x_1, x_2) \mid p_2(x_1, x_2) > 0, p_1(x_1, x_2) > 0\} \end{aligned}$$

[3]The proof of Lemma 4.1.1 serves as an algorithm for constructing such representations.

We proceed to the proof of the Partition Lemma.

Proof of the Partition Lemma for booleans on $\mathcal{R}$ (Lemma 4.1.1). By structural induction on the canonical form of booleans on $\mathcal{R}$.

Base case: $b \equiv p(\mathtt{x}) = 0$ or $p(\mathtt{x}) > 0$. Immediate from Lemma 3.3.5, because in each case there is a single basic set for each of the positive and negative sets respectively.

Induction step:

In what follows, suppose:

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

$NS(b_1) = \bigcup_{i=1}^{l_1} B_{1i}^-$

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

$NS(b_2) = \bigcup_{j=1}^{l_2} B_{2j}^-$

Now we consider the various cases based on the canonical form of b:

(i) $b \equiv \neg b_1$. Then just exchange the positive and negative sets of b_1; since all three properties hold for both, they still hold after switching.

(ii) $b \equiv b_1 \vee b_2$. Then

$$PS(b) = (\bigcup_{i=1}^{k_1}\bigcup_{j=1}^{k_2}(B_{1i}^+ \cap B_{2j}^+)) \cup (\bigcup_{i=1}^{k_1}\bigcup_{j=1}^{l_2}(B_{1i}^+ \cap B_{2j}^-)) \cup (\bigcup_{i=1}^{l_1}\bigcup_{j=1}^{k_2}(B_{1i}^- \cap B_{2j}^+))$$

$$NS(b) = \bigcup_{i=1}^{l_1}\bigcup_{j=1}^{l_2}(B_{1i}^- \cap B_{2j}^-)$$

The outer union of $PS(b)$ is disjoint, because $B_i^+ \cap B_j^- = \emptyset$. Further, the inner unions of $PS(b)$ and the union of $NS(b)$ are disjoint, because $B_{i_1}^+ \cap B_{i_2}^+ = \emptyset$ for $i_1 \neq i_2$, and $B_{j_1}^- \cap B_{j_2}^- = \emptyset$ for $j_1 \neq j_2$.

So $PS(b)$ and $NS(b)$ are finite unions of disjoint sets, all of which are basic, as the intersection of any two basic sets is also basic (Remark 3.3.4).

(iii) $b \equiv b_1 \wedge b_2$. Then

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

$$NS(b) = (\bigcup_{i=1}^{l_1} \bigcup_{j=1}^{l_2} (B_{1i}^{-} \cap B_{2j}^{-})) \cup (\bigcup_{i=1}^{k_1} \bigcup_{j=1}^{l_2} (B_{1i}^{+} \cap B_{2j}^{-})) \cup (\bigcup_{i=1}^{l_1} \bigcup_{j=1}^{k_2} (B_{1i}^{-} \cap B_{2j}^{+}))$$

Similar to case (ii), we observe that $PS(b)$ and $NS(b)$ are unions of disjoint basic sets.

(iv) $b \equiv b_1 \overset{c}{\vee} b_2$. Then

$$PS(b) = \bigcup_{i=1}^{k_1} B_{1i}^{+} \cup (\bigcup_{i=1}^{l_1} \bigcup_{j=1}^{k_2} (B_{1i}^{-} \cap B_{2j}^{+}))$$

$$NS(b) = \bigcup_{i=1}^{l_1} \bigcup_{j=1}^{l_2} (B_{1i}^{-} \cap B_{2j}^{-})$$

Again similar to case (ii).

(v) $b \equiv b_1 \overset{c}{\wedge} b_2$. Then

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

$$NS(b) = \bigcup_{i=1}^{k_1} B_{1i}^{-} \cup (\bigcup_{i=1}^{k_1} \bigcup_{j=1}^{l_2} (B_{1i}^{+} \cap B_{2j}^{-}))$$

Again similar to (ii).

□

Remark 4.1.4. The Partition Lemma for booleans on $\mathcal{R}$ (Lemma 4.1.1) does not hold for $\textbf{\textit{While}}^{\mathsf{OR}}$ or $\textbf{\textit{While}}^{\exists\mathsf{N}}$, where the '$\nabla$' operator is available, because, given any two booleans $b_1, b_2 \in \textbf{\textit{Bool}}(\mathtt{x})$, we cannot necessarily reduce the positive set of $b_1 \nabla b_2$ to a disjoint union, as we will see in section 4.3.

4.2 Structure Theorem for $\textbf{\textit{While}}(\mathcal{R})$ semicomputability

From [Fu14, §§4.6], we have the following lemma[4] for $\textbf{\textit{While}}(\mathcal{R})$ semicomputability:

Lemma 4.2.1. *For subsets of* $\mathbb{R}^2$*,*

(a) $\textbf{\textit{While}}(\mathcal{R})$ *s/comp* $\implies$ *countable union of effective disjoint sequence of* finite unions *of basic sets*

(b) *countable union of effective disjoint sequence* $\implies$ $\textbf{\textit{While}}(\mathcal{R})$ *s/comp. of basic sets*

We now strengthen this lemma by restricting the sequence in part *(a)* to basic sets rather than finite unions of basic sets, which provides us with an equivalence:

Theorem 1 (Structure Theorem for $\textbf{\textit{While}}(\mathcal{R})$)**.** *For subsets of* $\mathbb{R}^2$*,*

$\textbf{\textit{While}}(\mathcal{R})$ *s/comp* $\iff$ *countable union of an effective disjoint sequence of basic sets.*

Proof. The ' $\Longleftarrow$ ' direction is simply part *(b)* of Lemma 4.2.1

For the ' $\Longrightarrow$ ' direction, we strengthen part *(a)* of Lemma 4.2.1 as follows:

If $R \subseteq \mathbb{R}^2$ is $\textbf{\textit{While}}(\mathcal{R})$ semicomputable, then by Engeler's Lemma (Lemma 3.1.1), for all $x \in \mathbb{R}^2$,

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

for an effective sequence (b_k) of Σ-booleans in $\textbf{\textit{Bool}}(\mathbf{x})$. By the Partition Lemma, each b_k defines a finite union of effective disjoint basic sets.

[4] Lemma 4.2.1 was presented as a Structure Theorem for $\textbf{\textit{While}}(\mathcal{R})$ in [Fu14].

By the Semantic Disjointedness Lemma (Lemma 3.1.3), the sequence (b_k) is semantically disjoint over $\mathcal{R}$, and hence the positive sets for different b_k's are disjoint.

Hence b_k is a disjoint sequence of effective basic sets as desired. □

4.3 Failure of closure of $\textbf{\textit{While}}(\mathcal{R})$ semicomputable sets under union

For total standard algebras, we have the following lemma [TZ00, §§5.2], [TZ15, §§6.1]:

Lemma 4.3.1 (Closure of ***While*** semicomputable sets under union for total standard algebras)**.** *For any total standard algebra A, the class of* $\textbf{\textit{While}}(A)$ *semicomputable sets is closed under finite unions.*

We may use the Structure Theorem for $\textbf{\textit{While}}(\mathcal{R})$ (Theorem 1) to easily give a counterexample to the closure of semicomputable sets under finite union the partial algebra of the reals.

Theorem 2 (Failure of closure of $\textbf{\textit{While}}(\mathcal{R})$ semicomputable sets under union)**.** *The class of* $\textbf{\textit{While}}(\mathcal{R})$ *semicomputable sets on* $\mathbb{R}^2$ *is* not *closed under finite unions.*

For the proof, we require the following discussion:

Discussion 4.3.2 (A union of two basic sets which is not basic)**.** Consider the overlapping basic sets:

$$B_1 = \{(x_1, x_2) \mid -x_1^2 - x_2^2 + 1 > 0\},$$
$$B_2 = \{(x_1, x_2) \mid -(x_1 - 1)^2 - x_2^2 + 1 > 0\}$$

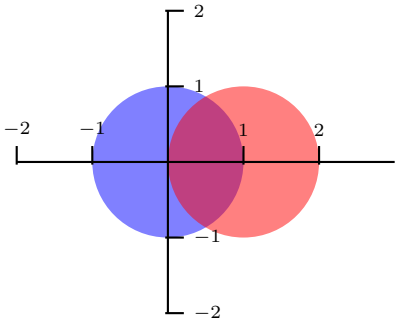
discussed in Discussion 4.1.3.

Their union is clearly semi-algebraic, but *not basic*.

This follows from a more general result from [ABR96]:
If the boundaries of two semi-algebraic subsets of $\mathbb{R}^2$ intersect transversally at some point, then their union is never basic[5].

Proof of Theorem 2.

Figure 4.5: $B_1 \cup B_2 = PS(b_1 \nabla b_2)$



Recall again the sets B_1 and B_2 from Discussion 4.1.3. Consider $B_1 \cup B_2 = PS(b_1 \nabla b_2)$, pictured in Figure 4.5. It is a union of two semicomputable sets (pictured in Figures 4.1 and 4.2). If it is semicomputable, then by the Structure Theorem for ***While*** it is an effective disjoint sequence of basic sets. However, since it is open and connected, it must in fact be equal to a single basic set. However, by Discussion 4.3.2, there is no way to represent $B_1 \cup B_2$ as a single basic sets. So while B_1 and B_2 are semicomputable, their union is not. □

Note that with respect to $\textbf{\textit{While}}^{\mathsf{OR}}(\mathcal{R})$ and $\textbf{\textit{While}}^{\exists\mathbb{N}}(\mathcal{R})$, the set $PS(b_1 \nabla b_2)$ is trivially semicomputable (cf. Remark 4.1.4).

[5] We thank Professor Bröcker (Münster) for clarifying this (personal communication).

Discussion 4.3.3. While it is intuitively clear that $B_1 \cup B_2$ is not basic, this fact still requires a proof, as outlined in Discussion 4.3.2. To underline this, we will now consider a brief example of a set which seems intuitively not basic, but in fact is.

Consider the basic sets:

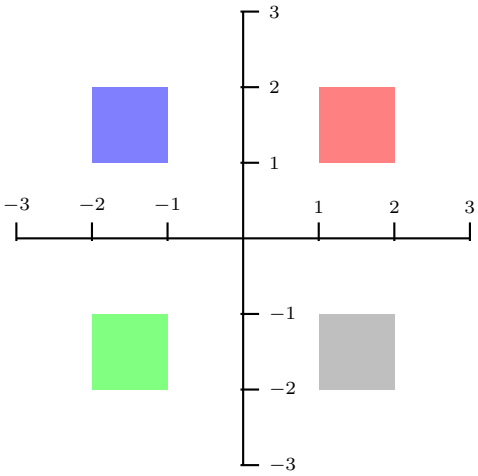
$$B_3 = PS(\{(x_1, x_2) \mid -2 < x_1 < -1 \ \wedge \ 1 < x_2 < 2\})$$

$$B_4 = PS(\{(x_1, x_2) \mid 1 < x_1 < 2 \ \wedge \ 1 < x_2 < 2\})$$

$$B_5 = PS(\{(x_1, x_2) \mid -2 < x_1 < -1 \ \wedge \ -2 < x_2 < -1\})$$

$$B_6 = PS(\{(x_1, x_2) \mid 1 < x_1 < 2 \ \wedge \ -2 < x_2 < -1\})$$

Figure 4.6: $B_3 \cup B_4 \cup B_5 \cup B_6$



Their union, pictured in Figure 4.6, seems intuitively not basic (though clearly it is semi-algebraic), if it is written in the obvious way:

$$\begin{aligned} B_3 \cup B_4 \cup B_5 \cup B_6 = PS(\{(x_1, x_2) \mid & (-2 < x_1 < -1 \ \wedge \ 1 < x_2 < 2) \\ & \vee (1 < x_1 < 2 \ \wedge \ 1 < x_2 < 2) \\ & \vee (-2 < x_1 < -1 \ \wedge \ -2 < x_2 < -1) \\ & \vee (1 < x_1 < 2 \ \wedge \ -2 < x_2 < -1)\}) \end{aligned}$$

However, the union is, in fact, basic, since it can be written as:

$$B_3 \cup B_4 \cup B_5 \cup B_6 = PS(\{(x_1, x_2) \mid 1 < x_1^2 < 4 \ \wedge \ 1 < x_2^2 < 4\})$$

Chapter 5

Classes of sets semicomputable by models based on the *While* language

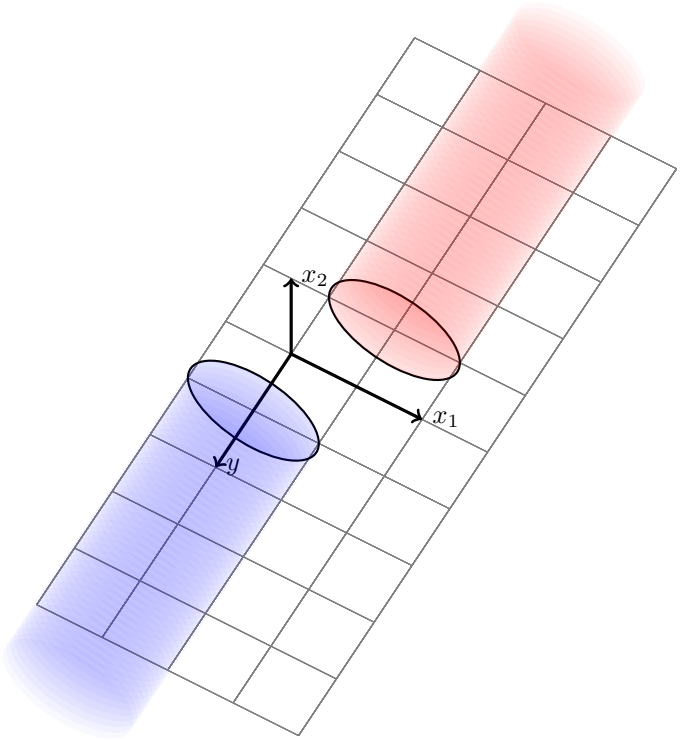
In this chapter we consider the equivalence or inequivalence of the classes of $\boldsymbol{While}(\mathcal{R})$, $\boldsymbol{While}^{\mathsf{OR}}(\mathcal{R})$ and $\boldsymbol{While}^{\exists\mathbb{N}}(\mathcal{R})$ semicomputable sets, as well as the semicomputable sets of the projective versions of those languages.

5.1 A set which is projectively $\boldsymbol{While}(\mathcal{R})$ semicomputable but not $\boldsymbol{While}(\mathcal{R})$ semicomputable

We will show that projective $\boldsymbol{While}(\mathcal{R})$ semicomputability is strictly stronger than $\boldsymbol{While}(\mathcal{R})$ semicomputability by giving a set which is projectively $\boldsymbol{While}(\mathcal{R})$

semicomputable but not ***While***$(\mathcal{R})$ semicomputable.

Figure 5.1: Domain of $f_0(x_1, x_2, y)$.



Consider the three-dimensional boolean valued function:

$$f_0(x_1, x_2, y) = \begin{cases} x_1^2 + x_2^2 < 1 & \text{if } y > 1 \\ (x_1 - 1)^2 + x_2^2 < 1 & \text{if } y < -1 \\ \uparrow & \text{o/w} \end{cases}$$

the domain of which is pictured in Figure 5.1.

The domain of f_0 is easily seen to be ***While***$(\mathcal{R})$ semicomputable, and so its projection off the third argument:

$$f(x_1, x_2) =_{df} \exists y : \mathbb{R}, (y < -1 \ \wedge \ x_1^2 + x_2^2 < 1) \vee (y > 1 \ \wedge \ (x_1 - 1)^2 + x_2^2 < 1)$$

is projectively ***While***$(\mathcal{R})$ semicomputable.

We have seen this set previously in Figure 4.5, and we have seen that it is not $\boldsymbol{While}(\mathcal{R})$ semicomputable (during the proof of Theorem 2), as it is not a union of disjoint basic sets.

From this example, we have the following theorem:

Theorem 3. *For subsets of* $\mathbb{R}^2$*,*

$$\boldsymbol{While}(\mathcal{R})\ \textit{s/comp} \underset{\not\Leftarrow}{\Rightarrow} \textit{proj-}\boldsymbol{While}(\mathcal{R})\ \textit{s/comp}$$

Proof. The ' $\Longrightarrow$ ' direction is clear from the definition of projective $\boldsymbol{While}(\mathcal{R})$ semicomputability. The ' $\Longleftarrow$ ' direction is clear from the above. □

Note that this set is $\boldsymbol{While}^{\mathsf{OR}}(\mathcal{R})$ semicomputable, from the following theorem given in [Fu14, §§4.6]:

Theorem (Structure Theorem for $\boldsymbol{While}^{\mathsf{OR}}(\mathcal{R})$)**.** *For subsets of* $\mathbb{R}^2$*,*

$\boldsymbol{While}^{\mathsf{OR}}(\mathcal{R})$ *s/comp* $\Longleftrightarrow$ *countable union of an effective disjoint sequence of* finite unions *of basic sets.*

Hence we can also compare $\boldsymbol{While}(\mathcal{R})$ and $\boldsymbol{While}^{\mathsf{OR}}(\mathcal{R})$ semicomputable sets:

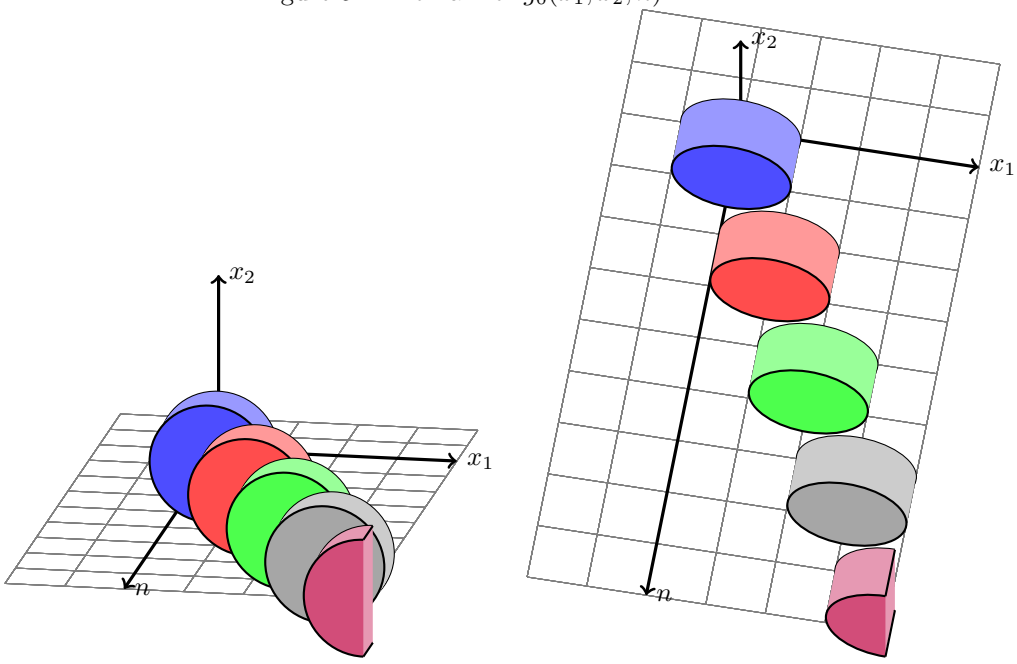
Theorem 4. *For subsets of* $\mathbb{R}^2$*,*

$$\boldsymbol{While}(\mathcal{R})\ \textit{s/comp} \underset{\not\Leftarrow}{\Rightarrow} \boldsymbol{While}^{\mathsf{OR}}(\mathcal{R})\ \textit{s/comp}$$

5.2 A set which is projectively $\boldsymbol{While}^{\mathsf{OR}}(\mathcal{R})$ semicomputable but not $\boldsymbol{While}^{\mathsf{OR}}$ semicomputable

We will now show a similar example which proves that projective $\boldsymbol{While}^{\mathsf{OR}}(\mathcal{R})$ semicomputability is stronger that $\boldsymbol{While}^{\mathsf{OR}}(\mathcal{R})$ semicomputability.

Figure 5.2: Domain of $g_0(x_1, x_2, n)$.



Consider the three dimensional boolean valued function:

$$g_0(x_1, x_2, n) = (x_1 - 2n)^2 + x_2^2 < 1$$

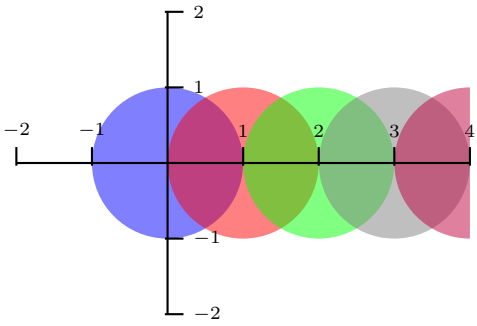
the domain of which is partially pictured from two viewpoints in Figure 5.2, for $x_1 \leq 4$. This function describes an infinite sequence of disjoint discs of depth 1 whose projections overlap in the x_1-x_2 plane.

Now, much as with the example of f_0 in section 5.1, we can easily observe the the domain of g_0 is $\textbf{\textit{While}}(\mathcal{R})$ semicomputable, and hence is also $\textbf{\textit{While}}^{\mathsf{OR}}(\mathcal{R})$ semicomputable. So the projection of its domain off the y axis:

$$g(x_1, x_2) =_{df} \exists n : \mathbb{N}, (x_1 - 2n)^2 + x_2^2 < 1$$

is projectively $\textbf{\textit{While}}^{\mathsf{OR}}(\mathcal{R})$ semicomputable.

Figure 5.3: Domain of $g(x_1, x_2)$.



The domain of g is partially pictured in Figure 5.3, again for $x_1 \leq 4$.

Now consider whether the domain of g is $\textbf{\textit{While}}^{\mathsf{OR}}(\mathcal{R})$ semicomputable. By the structure theorem for $\textbf{\textit{While}}^{\mathsf{OR}}(\mathcal{R})$ semicomputability (Theorem 5.1), that would mean its domain is an effective disjoint union of finite unions of basic sets. In this case, because the domain is connected, it must be a single semialgebraic set. By an analysis of semi-algebraic sets, we can see that this is not the case:

Discussion 5.2.1. Consider the domain of $g(x_1, x_2)$. Replace x_2 by 0.9, to get the one dimensional set:

$$\{x \in \mathbb{R} \mid g(x, 0.9) > 0\}.$$

If the domain of $g(x_1, x_2)$ is semi-algebraic, then this set should be semialgebraic.

However, this set is consists of infinitely many intervals (the intervals on which the horizontal line at $x_2 = 0.9$ intersects the "top" of the discs), and therefore is not semi-algebraic.[1]

So g is not semialgebraic, and therefore is not $\boldsymbol{While}^{\mathsf{OR}}(\mathcal{R})$ semicomputable. Therefore, we have:

Theorem 5. *For subsets of* $\mathbb{R}^2$*,*

$$\boldsymbol{While}^{\mathsf{OR}}(\mathcal{R})\ \textit{s/comp} \underset{\not\Leftarrow}{\Rightarrow} \textit{proj-}\boldsymbol{While}^{\mathsf{OR}}(\mathcal{R})\ \textit{s/comp}$$

Proof. The '$\Longrightarrow$' direction is clear from the definition of projective $\boldsymbol{While}^{\mathsf{OR}}(\mathcal{R})$ semicomputability. The '$\Longleftarrow$' direction is clear from the above. □

Note that this set is $\boldsymbol{While}^{\exists\mathbb{N}}(\mathcal{R})$ semicomputable, from the following theorem given in [Fu14, §§4.6]:

Theorem (Structure Theorem for $\boldsymbol{While}^{\exists\mathbb{N}}(\mathcal{R})$)**.** *For subsets of* $\mathbb{R}^2$*,*

$$\boldsymbol{While}^{\exists\mathbb{N}}(\mathcal{R})\ \textit{s/comp} \Longleftrightarrow \textit{countable union of an effective sequence of basic sets.}$$

Hence we can also compare $\boldsymbol{While}^{\mathsf{OR}}(\mathcal{R})$ and $\boldsymbol{While}^{\exists\mathbb{N}}(\mathcal{R})$ semicomputable sets:

Theorem 6. *For subsets of* $\mathbb{R}^2$*,*

$$\boldsymbol{While}^{\mathsf{OR}}(\mathcal{R})\ \textit{s/comp} \underset{\not\Leftarrow}{\Rightarrow} \boldsymbol{While}^{\exists\mathbb{N}}(\mathcal{R})\ \textit{s/comp}$$

5.3 Equivalence of projective $\boldsymbol{While}(\mathcal{R})$ and $\boldsymbol{While}^{\exists\mathbb{N}}(\mathcal{R})$ semicomputability

In [XFZ15, §§5.6], it was shown that $\boldsymbol{While}^{\exists\mathbb{N}}(\mathcal{R})$ semicomputability was equivalent to projective $\boldsymbol{While}^{\exists\mathbb{N}}(\mathcal{R})$ semicomputability:

[1] Thanks again to Prof. Bröcker for pointing this out.

Lemma 5.3.1. *For subsets of* $\mathbb{R}^2$*,*

$$\textit{proj-}\boldsymbol{While}^{\exists\mathsf{N}}(\mathcal{R})\ \textit{s/comp} \iff \boldsymbol{While}^{\exists\mathsf{N}}(\mathcal{R})\ \textit{s/comp}$$

Essentially, showing this involves replacing the projected arguments with auxiliary variables of sort nat which are existentially quantified over. Because of the continuity of $\boldsymbol{While}^{\exists\mathsf{N}}$ programs, quantifying over naturals suffices for replacing real projected arguments.

We will show that further, $\boldsymbol{While}^{\exists\mathsf{N}}(\mathcal{R})$ semicomputability is equivalent to projective $\boldsymbol{While}(\mathcal{R})$ semicomputability.

Lemma 5.3.2. *For subsets of* $\mathbb{R}^2$*,*

$$\textit{proj-}\boldsymbol{While}(\mathcal{R})\ \textit{s/comp} \iff \boldsymbol{While}^{\exists\mathsf{N}}(\mathcal{R})\ \textit{s/comp}$$

We need the following fact that follows definitions of the projective models:

Corollary 5.3.3. *For subsets of* $\mathbb{R}^2$*,*

$$\begin{aligned}\textit{proj-}\boldsymbol{While}(\mathcal{R})\ \textit{s/comp} &\implies \textit{proj-}\boldsymbol{While}^{\mathsf{OR}}(\mathcal{R})\ \textit{s/comp}\\ &\implies \textit{proj-}\boldsymbol{While}^{\exists\mathsf{N}}(\mathcal{R})\ \textit{s/comp}\end{aligned}$$

Proof. Follows from Theorem 4, Theorem 6 and the definition of the projective models. □

Remark 5.3.4. During the proof, we use the $\boldsymbol{While}^*$ language in place of the $\boldsymbol{While}$ language for simplicity. We require projection off an arbitrary but finite list of naturals. We can easily represent such a list using a single natural number argument for a $\boldsymbol{While}$ program, but to avoid tediousness we use the $\boldsymbol{While}^*$ language for the proof.

Proof of Lemma 5.3.2. The ' $\implies$ ' direction follows from Lemma 5.3.1 and Corollary 5.3.3.

For the ' $\Longleftarrow$ ' direction, consider any $\boldsymbol{While}^{\exists\mathbb{N}}(\mathcal{R})$ semicomputable set which is the halting set of a $\boldsymbol{While}^{\exists\mathbb{N}}$ progrem $P : \mathbb{R}^2$.[2]

We will construct a $\boldsymbol{While}^*$program $P_0 : \mathbb{R}^2 \times \mathbb{N}^*$ such that the projection of the domain of P_0 off of $\mathbb{N}^*$ is equal to the domain of P.

We construct P_0 from P by replacing each line of the form:

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

by the two lines:

$$\mathtt{x}^{\mathsf{B}} := P(t, \mathtt{z}[\mathrm{i}])$$
$$\mathtt{i} := \mathtt{i} + 1$$

where $\mathtt{i}$ is a new auxiliary variable which is initialized to 0 at the start of the program.

Then suppose that for some input values $r_1, r_2 : \mathbb{R}^2$, $P(r_1, r_2)$ halts. Then since P halted in finitely many steps, there exists a finite list of natural numbers $z_1, ..., z_n$ which are existentially quantified corresponding to the '$\mathtt{x}^b := \ \exists \mathtt{z}\ P(t, \mathtt{z})$' lines. This gives an array of naturals z such that $P_0(r_1, r_2, z)$ halts is given[3]. So the set $\boldsymbol{While}^{\exists\mathbb{N}}$ semicomputed by P is seen to also be semicomputable by a projective $\boldsymbol{While}^*$program and hence a $\boldsymbol{While}$ program (see Remark 5.3.4). □

5.4 Classes of sets semicomputable by models based on the *While* language

We now consider the classes of sets semicomputable by the $\boldsymbol{While}$, $\boldsymbol{While}^{\mathsf{OR}}$ and $\boldsymbol{While}^{\exists\mathbb{N}}$ languages and their projective versions.

[2] As with all of our (main) results, we restrict our attention to functions on $\mathbb{R}^2$. Note that this proof in particular is easily extendable to functions of any type.

[3] Note that the order of the naturals used in the existential quantification steps may have little relation to the order of the $\mathtt{x}^b := \ \exists \mathtt{z}\ \mathrm{P(t, z)}$ lines in the code, due to loops and branches.

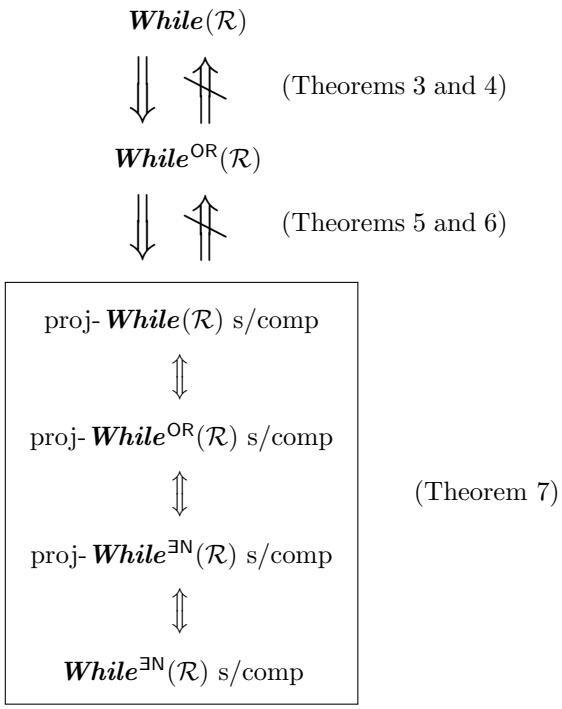
We begin by combining the results discussed in the previous section:

Theorem 7. *For subsets of* $\mathbb{R}^2$,

$$\begin{aligned} \textit{proj-}\boldsymbol{While}(\mathcal{R})\ \textit{s/comp} &\iff \textit{proj-}\boldsymbol{While}^{\mathsf{OR}}(\mathcal{R})\ \textit{s/comp} \\ &\iff \textit{proj-}\boldsymbol{While}^{\exists\mathbb{N}}(\mathcal{R})\ \textit{s/comp} \\ &\iff \boldsymbol{While}^{\exists\mathbb{N}}(\mathcal{R})\ \textit{s/comp} \end{aligned}$$

Proof. Follows from Lemma 5.3.1, Lemma 5.3.2 and Corollary 5.3.3. □

We have thus established the existence of three distinct classes of subsets of $\mathbb{R}^2$, as shown in the following diagram:



We further have the equivalence of each model in the above diagram with its respective starred version (as shown in Appendix A).

Chapter 6

Conclusion and future work

6.1 Conclusion

In this thesis, we investigated the generalisation of two results from classical computability theory to the context of topological algebra of the reals: closure of semicomputable sets under union and the equivalence semicomputable sets of projectively (semi)computable sets. Both results were shown to not hold in the context of the reals (Theorem 2 and Theorem 3 respectively).

In the process we also developed a Structure Theorem for $\boldsymbol{While}(\mathcal{R})$ semicomputability (Theorem 1), and distinguished the classes of sets semicomputed by $\boldsymbol{While}(\mathcal{R})$, $\boldsymbol{While}^{\mathsf{OR}}(\mathcal{R})$ and $\boldsymbol{While}^{\exists\mathsf{N}}(\mathcal{R})$ programs and their projective versions (section 5.4).

In Appendix B, we show that another result from classical computability theory, Post's Theorem, holds trivially in the case of $\mathcal{R}$, but does not hold more generally for partial algebras.

6.2 Future work

We have compared various classes of subsets of $\mathbb{R}^2$ by abstract models based on the ***While*** language (section 5.4). Similarly, we would like to investigate concrete models of computability (section 1.1), and compare them amongst themselves and with abstract models. In [TZ04, TZ05], an equivalence was found between certain abstract and concrete models. In [Fu14], several such concrete models were shown to be equivalent.

An open problem in this area is the relationship of models considered in the above papers with Weihrauch's TTE (type two effective) model of computation [Wei00].

The investigation of this problem is intended for a major part of the writer's PhD thesis.

Bibliography

[ABR96] C. Andradas, L. Bröcker, and J. Ruiz. *Constructible Sets in Real Geometry*. Springer, 1996.

[Chu36] A. Church. An unsolvable problem of elementary number theory. *—amer— Journal of Mathematical—*, 58:345–363, 1936.

[Eng68] E. Engeler. *Formal Languages: Automata and Structures*. Markham, 1968.

[Fu14] Ming Quan Fu. *Characterizations of Semicomputable Sets, and Computable Partial Functions, on the Real Plane*. PhD Thesis, Department of Computing & Software, McMaster University, 2014. Archived in DSpace at http://hdl.handle.net/11375/16066.

[Grz55] A. Grzegorczyk. Computable functions. *Fundamenta Mathematicae*, 42:168–202, 1955.

[Grz57] A. Grzegorczyk. On the defintions of computable real continuous functions. *Fundamenta Mathematicae*, 44:61–71, 1957.

[Kle36] S.C. Kleene. General recursive functions of natural numbers. *Mathematische Annalen*, 112:727–742, 1936.

[Kle52] S.C. Kleene. *Introduction to Metamathematics*. North Holland, 1952.

[Tur36] A.M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936. With correction, *ibid.*, 43, 544–546, 1937. Reprinted in *The Undecidable*, M. Davis, ed., Raven Press, 1965.

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

[TZ00] J.V. Tucker and J.I. Zucker. Computable functions and semicomputable sets on many-sorted algebras. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 5, pages 317–523. Oxford University Press, 2000.

[TZ04] J.V. Tucker and J.I. Zucker. Abstract versus concrete computation on metric partial algebras. *ACM Transactions on Computational Logic*, 5:611–668, 2004.

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

[TZ15] J.V. Tucker and J.I. Zucker. Generalizing computability theory to abstract algebras. In G. Sommaruga and T. Strahm, editors, *Turing's Revolution. The Impact of his Ideas about Computability*, page (To appear). Birkhauser/Springer Basel, 2015.

[Wei00] K. Weihrauch. *Computable Analysis: An Introduction*. Springer, 2000.

[XFZ15] Bo Xie, Ming Quan Fu, and Jeffery Zucker. Characterizations of semicomputable sets of real numbers. *Journal of Logic and Algebraic Programming*, 84:124–154, 2015.

Appendix A

The equivalence of $\boldsymbol{While}(\mathcal{R})$ and $\boldsymbol{While}^*(\mathcal{R})$

We wish to justify our claims that the $\boldsymbol{While}(\mathcal{R})$ and $\boldsymbol{While}^*(\mathcal{R})$ are equivalent in terms of computing power.

A similar result was shown in [TZ00, §4] for a *total* algebra of the reals, by showing that:

(1) the total algebra of the reals has the "*term evaluation property*" (defined below)

(2) for any $\mathbb{N}$-standard total algebra A with the term evaluation property, a universal $\boldsymbol{While}$ procedure may be constructed for $\boldsymbol{While}^*$

(3) hence, for any $\mathbb{N}$-standard total algebra A with the term evaluation property, $\boldsymbol{While}(A) = \boldsymbol{While}^*(A)$.

We may use the same process to show that $\boldsymbol{While}(\mathcal{R}) = \boldsymbol{While}^*(\mathcal{R})$ (and similarly for $\boldsymbol{While}^{\mathsf{OR}}$ and $\boldsymbol{While}^{\exists\mathbb{N}}$). Steps (2) and (3) can be easily inferred from the respective proofs in [TZ00, §4], as most of the proofs of those facts

involve primitive recursive operations on the syntax of the ***While*** language, and so the partiality of $\mathcal{R}$ is irrelevant. In these proofs, the only step during the proofs that involves semantics is the use of term evaluation to traverse a "computation tree" for the universal $\boldsymbol{While}(A)$ procedure for $\boldsymbol{While}^*(A)$ programs. In that step, however, if a term which is evaluated diverges, the $\boldsymbol{While}^*(A)$ program being simulated by the universal procedure would diverge as well, and so the universal procedure behaves as expected.

So we proceed with a proof of Step (1), i.e. that $\mathcal{R}$ has the term evaluation property (cf. [TZ00, Example 4.5]). Note that the partiality of $\mathcal{R}$ presents no problems during the proof; it simply means that the term evaluation function must be partial.

For the remainder of this Appendix, let A be any $\mathbb{N}$-standard (possibly partial) algebra, let u and v be product types of A, let $\mathtt{x}$ be a u-tuple of variables, let $\boldsymbol{Tm}_{\mathtt{x}}(\Sigma)$ be the set of all Σ-terms with variables among $\mathtt{x}$ only, and for all sorts s or Σ, let $\boldsymbol{Tm}_{\mathtt{x},s}(\Sigma)$ be the class of such terms of sort s.

We define this *term evaluation function on A relative to* $\mathtt{x}$

$$\boldsymbol{TE}^A_{\mathtt{x},s} : \boldsymbol{Tm}_{\mathtt{x},s} \times \boldsymbol{State}(A) \to A_s$$

by

$$\boldsymbol{TE}^A_{\mathtt{x},s}(t, \sigma) \simeq [\![t]\!]^A\sigma.$$

We represent the term evaluation function on A on relative to $\mathtt{x}$ by the function

$$\boldsymbol{te}^A_{\mathtt{x},s} : \ulcorner \boldsymbol{Tm}_{\mathtt{x},s} \urcorner \times A^u \to A_s$$

defined by

$$\boldsymbol{te}^A_{\mathtt{x},s}(\ulcorner t \urcorner, a) \simeq [\![t]\!]^A\sigma,$$

where σ is any state on A such that $\sigma[\mathtt{x}] = a$ (this is well defined, by Lemma 3.4 from [TZ00]).

Definition A.0.1. An algebra A has the *term evaluation property* (*TEP*) if for all $\mathtt{x}$ and s, $\boldsymbol{te}^{A}_{\mathtt{x},s}$ is $\boldsymbol{While}(A)$ computable.

Lemma A.0.2. *$\mathcal{R}$ has the TEP.*

Proof (outline).
We may give the following primitive recursive algorithm which shows the computability of $\boldsymbol{te}^{\mathcal{R}}_{\mathtt{x},s}$; for instance, in the case that $s \equiv \mathsf{bool}$:

$$\begin{aligned}
\boldsymbol{te}^{\mathcal{R}}_{\mathtt{x},\mathsf{bool}}(\ulcorner t_s \ \mathsf{comp}\ r_s \urcorner, a) &\simeq \boldsymbol{te}^{\mathcal{R}}_{\mathtt{x},s}(\ulcorner t_s \urcorner, a)\ \mathsf{comp}\ \boldsymbol{te}^{\mathcal{R}}_{\mathtt{x},s}(\ulcorner r_s \urcorner, a) \\
\boldsymbol{te}^{\mathcal{R}}_{\mathtt{x},\mathsf{bool}}(\ulcorner \mathsf{not}^{\mathsf{B}}(b_1) \urcorner, a) &\simeq \mathsf{not}^{\mathsf{B}}(\boldsymbol{te}^{\mathcal{R}}_{\mathtt{x},\mathsf{bool}}(\ulcorner b_1 \urcorner, a)) \\
\boldsymbol{te}^{\mathcal{R}}_{\mathtt{x},\mathsf{bool}}(\ulcorner b_1 \ \mathsf{op}\ b_2 \urcorner, a) &\simeq \boldsymbol{te}^{\mathcal{R}}_{\mathtt{x},\mathsf{bool}}(\ulcorner b_1 \urcorner, a)\ \mathsf{op}\ \boldsymbol{te}^{\mathcal{R}}_{\mathtt{x},\mathsf{bool}}(\ulcorner b_2 \urcorner, a)
\end{aligned}$$

where sort s is either nat or real, b_1 and b_2 are terms of sort bool, comp is a comparison operator on s (one of eq^{N}, $\mathsf{less}^{\mathsf{N}}$, eq^{R} or $\mathsf{less}^{\mathsf{R}}$), and op is a binary boolean operator.

We omit the corresponding definitions for the cases $s \equiv \mathsf{real}$ and $s \equiv \mathsf{nat}$. □

The equivalence of $\boldsymbol{While}(\mathcal{R})$ and $\boldsymbol{While}^{*}(\mathcal{R})$ now follows from the above Lemma and the preceeding discussion.

Appendix B

A counterexample to Post's Theorem for partial algebras

For total standard algebras, we have the following theorem [TZ00, §§5.2], [TZ15, §§6.1]:

Theorem 8 (Post's theorem for ***While*** semicomputability on total algebras)**.** *For any relation R on any total algebra A,*

$$R \text{ is } \textbf{\textit{While}}(A) \text{ computable} \iff R \text{ and } R^c \text{ are } \textbf{\textit{While}}(A) \text{ semicomputable.}$$

We will show that Post's Theorem also holds on the partial algebra $\mathcal{R}$, but that there are some partial algebras on which it does not hold.

Theorem 9 (Post's theorem for $\textbf{\textit{While}}(\mathcal{R})$ semicomputability)**.** *For any relation R on $\mathcal{R}$,*

$$R \text{ is } \textbf{\textit{While}}(\mathcal{R}) \text{ computable} \iff R \text{ and } R^c \text{ are } \textbf{\textit{While}}(\mathcal{R}) \text{ semicomputable.}$$

Proof. The ' $\Longrightarrow$ ' direction is obvious; if R is computable, then R^c is also computable and hence semicomputable.

For the ' $\Longleftarrow$ ' direction, recall that by the Structure Theorem for $\boldsymbol{While}(\mathcal{R})$, if R and R^c are $\boldsymbol{While}(\mathcal{R})$ semicomputable then they are unions of effective disjoint sequences of basic sets, and recall that those basic sets are open. Then since any union of open sets is open, both R and R^c are open, and since the complement of an open set is closed, they are in fact both clopen. Then since the only clopen sets of the reals are the empty set and $\mathbb{R}^2$, R is one of those two, and is clearly computable. □

We now give an example of a standard partial algebra[1] $\mathcal{D}$ such that Post's theorem does not hold for its $\boldsymbol{While}$ semicomputable subsets.

Define a signature $\Sigma^{\mathcal{D}}$:

signature	$\Sigma^{\mathcal{D}}$
sorts	$\mathsf{data}, \mathsf{bool}$
functions	$0 : \ \to \mathsf{data}$
	$\mathsf{F}, \mathsf{G} : \mathsf{data} \rightharpoonup \mathsf{data}$
	$\mathsf{eq} : \mathsf{data}^2 \to \mathsf{bool}$

and algebra $\mathcal{D}$:

algebra	$\mathcal{D}$
carriers	$\mathbb{D}, \mathbb{B}$
functions	$0 : \ \to \mathbb{D}$
	$\mathsf{F}^{\mathsf{D}}, \mathsf{G}^{\mathsf{D}} : \mathbb{D} \rightharpoonup \mathbb{D}$
	$= : \mathbb{D}^2 \to \mathbb{B}$

where $\mathbb{D} = \{a, b, 0\}$ is the carrier for sort data and F^{D} and G^{D} are defined as follows:

[1]The proof for an $\mathbb{N}$-standard algebra is a routine, but tedious, extension. To see this observe that simply adding the naturals to the algebra gives us no additional computing power with regards to the data sort.

$$\mathsf{F}^{\mathsf{D}}(x) = \begin{cases} x & \text{if } x = a \\ \uparrow & \text{o/w} \end{cases}$$

$$\mathsf{G}^{\mathsf{D}}(x) = \begin{cases} x & \text{if } x \neq a \\ \uparrow & \text{o/w} \end{cases}$$

and $=^{\mathsf{D}}$ is the equality operator for sort $\mathbb{D}$.

Remark B.0.1. Any composition of the F^{D} and G^{D} functions is eliminable, in that for any term t:

- $\mathsf{F}^{\mathsf{D}}(\mathsf{F}^{\mathsf{D}}(t)) \simeq \mathsf{F}^{\mathsf{D}}(t)$
- $\mathsf{G}^{\mathsf{D}}(\mathsf{G}^{\mathsf{D}}(t)) \simeq \mathsf{G}^{\mathsf{D}}(t)$
- $\mathsf{F}^{\mathsf{D}}(\mathsf{G}^{\mathsf{D}}(t)) \uparrow$
- $\mathsf{G}^{\mathsf{D}}(\mathsf{F}^{\mathsf{D}}(t)) \uparrow$.

We will prove:

Proposition B.0.2. *Any boolean valued* ***While***$(\Sigma^{\mathcal{D}})$ *program which halts on inputs a and b must give the same output in both cases.*[2]

Remark B.0.3. It follows from Proposition B.0.2 that $\{a\}$ and $\{a\}^c = \{b, 0\}$ are both clearly semicomputable, but neither is computable.

In preparation for the proof of Proposition B.0.2, we need two related notions: *compatible set of states* and *set of distinguished variables.*

Definition B.0.4 (Compatible states; Distinguished variables)**.**
Let $\underline{A}$ be a Σ-structure and s a Σ-sort. Suppose for some $n \geq 2$, we have:

- n states $\sigma_1, ..., \sigma_n$,

[2]It is certainly possible for a data valued program to give different outputs on a and b; consider the identity function.

- n *distinct* elements $a_1, ..., a_n$ of A_s and
- a *finite* set V of variables of sort s, such that

$$\forall \mathtt{x} \notin V,\ \sigma_1(\mathtt{x}) = ... = \sigma_n(\mathtt{x})$$

and

$$\forall \mathtt{x} \in V,\ \sigma_1(\mathtt{x}) = a_1, ..., \sigma_n(\mathtt{x}) = a_n.$$

We call $\{\sigma_1, ..., \sigma_n\}$ a *compatible set of states with respect to* V.

We call V the *set of distinguished variables* for these states.

Note that the set of distinguished variables is unique for any particular set of compatible states.

Now (returning to the special case of $\underline{A} = \mathcal{D}$, $s = \mathsf{data}$) take two compatible states σ_a and σ_b over $\mathcal{D}$ with some set of distinguished variables V such that for all $\mathtt{v} \in V$

$$\sigma_a(\mathtt{v}) = a, \quad \sigma_b(\mathtt{v}) = b.^3$$

For the remainder of the section, we let S be any statement in $\boldsymbol{While}(\Sigma^{\mathcal{D}})$ for which the following assumption holds:

Assumption B.0.5. *S halts on σ_a and σ_b.*

In order to state and prove Lemma B.0.6 below, from which Proposition B.0.2 will follow, we need to reason about the snapshot sequences (recall the definition in section 2.9) generated by S at σ_a and σ_b.

We write:

$$(\boldsymbol{\sigma}_a, \boldsymbol{S}) = (\sigma_a^0, S_a^0), (\sigma_a^1, S_a^1), (\sigma_a^2, S_a^2), ...$$
$$(\boldsymbol{\sigma}_b, \boldsymbol{S}) = (\sigma_b^0, S_b^0), (\sigma_b^1, S_b^1), (\sigma_b^2, S_b^2), ...$$

[3] For instance, the states which assign a and b respectively to all $\mathtt{v} \in V$ and $\mathtt{0}$ to every other variable are good candidates for σ_a and σ_b.

We may now state Lemma B.0.6:

Lemma B.0.6. *The snapshot sequences* $(\boldsymbol{\sigma}_a, \boldsymbol{S})$*, and* $(\boldsymbol{\sigma}_b, \boldsymbol{S})$ *are isomorphic in that for all* n*,* $S_a^n \equiv S_b^n$*, and* σ_a^n *and* σ_b^n *are compatible with a set* V^n *of distinguished variables which are assigned* a *and* b *when the starting states are* σ_a *and* σ_b *respectively.*

In fact, the set V^n can be effectively identified given any σ_a, σ_b and n.

Lemma B.0.6 says, in effect, that the computations of S on σ_a and σ_b are "essentially the same". From this, Proposition B.0.2 follows.

The following Remark is central to the induction step of the proof of Lemma B.0.6:

Remark B.0.7. Any subterm of a term which is evaluated at stage n of the computation of S at σ_a or σ_b cannot have the form $\mathsf{F}(\mathsf{y})$ or $\mathsf{G}(\mathsf{y})$ where y is a distinguished variable of the states σ_a^n and σ_b^n, since that would violate Assumption B.0.5.[4,5] This essentially restricts our tests on the input variable to boolean combinations of tests for equality; specifically, due to the lack of closed terms for a and b, equality between variables or tests for zero.

The fact that tests on the input are limited to tests for equality between variables and tests for zero will allow us to show that, restricting our attention to states for which all variables of sort data are assigned the same values (such as σ_a and σ_b) and statements S for which Assumption B.0.5 holds, branching decisions on input states which only vary on their data values are the same between those states, and can be trivially predetermined.

We now prove Lemma B.0.6 by giving an algorithm to find the compatible variable set for σ_a^n and σ_b^n for any n.

[4] Recall by Remark B.0.1 that we can eliminate occurrences of F and G from other contexts in S.

[5] The existance of compatible states and distinguished variables at stage n is given by Lemma B.0.6

Proof of Lemma B.0.6. By course of values induction on the computation length of S at σ_a.[6]

Assume that for n, Lemma B.0.6 holds for any S such that $\boldsymbol{CompLen}^D(S, \sigma_a) < n$. We show that it also holds for S such that $\boldsymbol{CompLen}^D(S, \sigma_a) = n$. There are 5 cases to consider based on the form of S:

(1) $S \equiv \mathsf{skip}$: Trivial (compare with (2)).

(2) $S \equiv \mathtt{x} := t$: By $\boldsymbol{CompLen}^D(S, \sigma_a) = n$, we have $n = 1$, and so $\boldsymbol{Comp}^D(S, \sigma, n) = \langle |S| \rangle^D = \sigma\{\mathtt{x}/[\![t]\!]^D\sigma\}$.[7] Consider sub-cases based on t:[8]

  (i) $t \equiv 0$: Then take $V' = V \backslash \{\mathtt{x}\}$, and observe that V' is an appropriate set of distinguished variables.

  (ii) $t \equiv \mathtt{y}$: Then if $\mathtt{y} \in V$, take $V' = V \cup \{\mathtt{x}\}$, and otherwise take $V' = V$. Then observe that V' is an appropriate set of distinguished variables .

  (iii) $t \equiv \mathsf{F}^{\mathsf{D}}(\mathtt{y})$ where $\mathtt{y}$ is not a distinguished variable of σ_a, σ_b, and σ_c : then take $V' = V \backslash \{\mathtt{x}\}$, and observe that V' is an appropriate set of distinguished variables.

  (iv) $t \equiv \mathsf{G}^{\mathsf{D}}(\mathtt{y})$ where $\mathtt{y}$ is not a distinguished variable of σ_a, σ_b, and σ_c : similar to (iii).

  (v) $t \equiv \mathsf{F}^{\mathsf{D}}(\mathtt{y})$ where $\mathtt{y}$ is a distinguished variable of σ_a : by Remark B.0.7, such a term cannot appear in S.

  (vi) $t \equiv \mathsf{G}^{\mathsf{D}}(\mathtt{y})$ where $\mathtt{y}$ is a distinguished variable of σ_a : similar to (v).

  (vii) $t \equiv \mathsf{F}^{\mathsf{D}}(0)$: by Assumption B.0.5, since $\mathsf{F}^{\mathsf{D}}_a(0)$ diverges at all states, such a term cannot appear in S.

  (viii) $t \equiv \mathsf{G}^{\mathsf{D}}(0)$: similar to (vii).

[6] It will follow from the proof that the computation length of S at σ_a is the same as the computation length of S at σ_b.

[7] By the definition of $\boldsymbol{Comp}^D$ given in [TZ00, §§3.4].

[8] Recall by Remark B.0.1 that we may limit the use of F_a and F_b to this finite set of cases.

So for each which can appear in S, we have that σ_a^n and σ_b^n are compatible with respect to some V'. We also clearly have that $S_a^n \equiv S_b^n \equiv \mathsf{skip}$.

(3) $S \equiv S_1; S_2$: Let $\boldsymbol{CompLen}^D(S_1, \sigma_a) = k < n$ and $\boldsymbol{CompLen}^D(S_2, \sigma_a^k) = l = n - k$. By the induction hypothesis, since $i < n$, $S_a^i \equiv S_b^i$ and σ_a^i and σ_b^i are compatible with respect to a set of variables V_1. Now consider the snapshot sequences $(\boldsymbol{\sigma_a^i}, \boldsymbol{S_a^i})$ and $(\boldsymbol{\sigma_b^i}, \boldsymbol{S_b^i})$. By the induction hypothesis, since $j < n$, $S_a^{ij} \equiv S_b^{ij}$ and σ_a^{ij} and σ_b^{ij} are compatible with respect to a set of variables V_2. Now note that $S_a^n \equiv S_a^{ij}$ and $S_b^n \equiv S_b^{ij}$, and $\sigma_a^{ij} = \sigma_a^n$ and $\sigma_b^{ij} = \sigma_b^n$. So by taking $V' = V_2$, we have that $S_a^n \equiv S_b^n$ and σ_a^n and σ_b^n are compatible with respect to V'.

(4) $S \equiv \mathsf{if}\ t^{\mathsf{B}}\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}$: Note that the only boolean test for A is $=$, so t_b is a combination of tests of equality between variables or tests for zero[9]. It is easy to see that $[\![t_b]\!]^D\sigma_a = [\![t_b]\!]^D\sigma_b$.[10] Now our induction hypothesis holds for the snapshot sequences $(\boldsymbol{\sigma}_a, \boldsymbol{S_1})$, $(\boldsymbol{\sigma}_b, \boldsymbol{S_1})$, $(\boldsymbol{\sigma}_a, \boldsymbol{S_2})$ and $(\boldsymbol{\sigma}_b, \boldsymbol{S_2})$. And for the case that $[\![t_b]\!]^D\sigma_a = \mathsf{true}$, it is clear it also holds for the snapshot sequences

$$(\boldsymbol{\sigma}_a, \boldsymbol{S}) = (\sigma_a, S), (\boldsymbol{\sigma}_a, \boldsymbol{S_1})$$

$$(\boldsymbol{\sigma}_b, \boldsymbol{S}) = (\sigma_b, S), (\boldsymbol{\sigma}_b, \boldsymbol{S_1})$$

The case that $[\![t^{\mathsf{B}}]\!]^D\sigma_a = \mathsf{false}$ is similar.

(5) $S \equiv \mathsf{while}\ t^{\mathsf{B}}\ \mathsf{do}\ S_0\ \mathsf{od}$: As in (4), we have that t_b is a combination of zero tests or equality between variables, and $[\![t_b]\!]^D\sigma_a = [\![t_b]\!]^D\sigma_b$. Now our induction hypothesis holds for the snapshot sequence $(\boldsymbol{\sigma}_a, \boldsymbol{S_0})$. And for the case $[\![t_b]\!]^D\sigma_a = \mathsf{true}$, it must also hold for the snapshot sequences

[9] See Remark B.0.7.

[10] By assumption B.0.5, t_b does not diverge on any state.

$$(\boldsymbol{\sigma}_a, \boldsymbol{S}) = (\sigma_a, S), (\boldsymbol{\sigma}_a, \boldsymbol{S_0; S})$$

$$(\boldsymbol{\sigma}_b, \boldsymbol{S}) = (\sigma_b, S), (\boldsymbol{\sigma}_b, \boldsymbol{S_0; S})$$

because although $S_0; S$ is a more complex program that S, by Assumption B.0.5, the while loop must halt and so its computation length must be less, hence we can apply the induction hypothesis.

The case that $[\![t_b]\!]^D \sigma_a = \mathsf{false}$ is similar.

□

This proves Lemma B.0.6 and hence Proposition B.0.2, thus providing (by Remark B.0.3) a counterexample to Posts Theorem on the algebra $\mathcal{D}$.