WHILECC-APPROXIMABILITY AND ACCEPTABILITY OF ELEMENTARY FUNCTIONS

WHILECC-APPROXIMABILITY AND ACCEPTABILITY OF ELEMENTARY FUNCTIONS

By FATEME GHASEMI B.Sc.

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

Department of Computing and Software
McMaster University

Master of Science (2025)
(Computing and Software)

McMaster University
Hamilton, Ontario, Canada

TITLE:	***WhileCC***-approximability and Acceptability of Elementary Functions
AUTHOR:	Fateme Ghasemi, B.Sc. (University of Tehran)
SUPERVISOR:	Professor Jeffery I. Zucker
NUMBER OF PAGES:	vii, 79

Lay Abstract

Several models of computability were previously proposed for partial functions over the reals. Some of these models were proved to be equivalent for functions satisfying specific conditions we call "acceptability". In this thesis, we prove that at least the class of elementary functions satisfies this "acceptability" condition. This shows that the acceptability conditions are sufficiently general.

Abstract

In this thesis, we study models of computation for partial functions on the reals.

Existing work [Fu and Zucker, 2014, Tucker and Zucker, 1999, 2004] studies classes of computable partial functions on $\mathbb{R}$, namely

- GL-computability,
- tracking computability,
- multipolynomial approximability, and
- ***WhileCC***-approximability.

Fu and Zucker [2014] show that all these four models of computation are equivalent when we restrict our attention to a specific class of functions we call "acceptable" functions. This means, within the realm of acceptable functions, we can work with ***WhileCC***-approximability without giving up expressivity and transfer results amongst the models.

However, it was previously unknown whether the class of acceptable functions is sufficiently large to include many common functions, such as the elementary functions. In this thesis, we solve the conjecture posed by Fu and Zucker [2014] and show that all elementary functions are acceptable. We also prove that the elementary functions are ***WhileCC***-approximable and therefore computable in all the aforementioned models of computation.

Acknowledgements

Honestly, I couldn't have made it through this without some incredible people in my life – it truly took a village to write this thesis. Thank you all so much!

First and foremost, my deepest thanks go to Dr. Jeffery Zucker and Dr. Wolfram Kahl.

Dr. Zucker, thank you for all your help and support during my studies. Without your belief in me, I wouldn't have made it here – quite literally. You trusted me when I wasn't even sure I could trust myself, and because of that, I'm in a much better place today. And a special thank you to your wife, Shirley, for her warmth and kindness throughout the years – it never went unnoticed.

Dr. Kahl, you have been there for me since the start of my degree, always encouraging and helping me in every way imaginable. Thank you for being someone I could always count on! I also truly appreciate your careful and thorough review of my thesis draft and the valuable comments you provided. I seriously couldn't have done this without you. I will always be grateful.

Anthony, my dearest, I honestly can't imagine getting through this without you. You've truly been my rock every step of the way. I'm beyond grateful for everything you've done. Also, a huge thank you to your parents, Sheila and George – your kindness, support, and love made such a difference, and I'm deeply thankful.

To my brothers and my dad, thank you for always being there for me. Your support, your check-ins, and just knowing you were cheering me on from a distance meant more than you know. Sorry for all the "*Can't talk, still working on my thesis...*" messages – you know I still love you guys, right?

To my closest friends, Anana (yes, her name is Anahita, but I like to call her Anana, as in a pineapple!), Ghazal, and Alireza – thank you for being there through every meltdown, crisis, and weird late-night rant. You've seen me at my worst and still pushed me forward. I'm so lucky to have had you all in my corner.

To my manager at Huawei, Dr. Bingzhou Zheng, thank you for your understanding, support, and flexibility during my co-op – especially around the time of my back surgery. Your support made a difficult time much easier, and I truly appreciate it.

Finally, a huge thank you to Dr. Franya Franek and (again) Dr. Wolfram Kahl for being on my committee, carefully reading my thesis, providing thoughtful feedback and detailed suggestions, and sitting through my defense with grace (and minimal yawning). I truly appreciate your time and support.

Contents

List of Figures

Chapter 1

Introduction

In this thesis, we study models of computation for the reals. Previous work [Stoltenberg-Hansen and Tucker, 1999, Tucker and Zucker, 2005] focuses on computational models for total functions. However, the class of total functions is too restrictive: many standard functions in real analysis (such as the logarithmic, square root, and inverse trigonometric functions) are partial and cannot be studied under such models. Thus, in this thesis, we focus on the computability of partial functions.

Existing work [Fu and Zucker, 2014, Tucker and Zucker, 1999, 2004] studies classes of computable partial functions on $\mathbb{R}$, namely

- GL-computability,
- tracking computability,
- multipolynomial approximability, and
- **WhileCC**-approximability.

The first two classes correspond to concrete models of computation. In concrete models, computability depends on the representation of data. For example, an α-tracking computable function represents each real number as a natural number. In contrast, abstract models (such as **WhileCC**-programs) allow functions to be defined independently of such implementation details. For a programmer, this would be akin to writing programs against an abstract interface instead of dealing with specific implementations. Abstract models are easier to program in, but may not be as expressive as their concrete counterparts.

Fu and Zucker [2014] show that all these four models of computation are equivalent when we restrict our attention to a specific class of functions we call "acceptable" functions. This means, within the realm of acceptable functions, we can work with **WhileCC**-approximability without giving up expressivity and transfer results amongst the models.

However, it has been unknown whether the class of acceptable functions is sufficiently large to include many common functions, such as the elementary functions. In this thesis, we solve the conjecture posed by Fu and Zucker [2014] and show that all elementary functions are acceptable. We also prove that the elementary functions are **WhileCC**-approximable and therefore computable in all the aforementioned models of computation.

The contributions of this thesis are as follows:

- We prove that all elementary functions are **WhileCC**-approximable.

- We prove that all elementary functions are acceptable.

- We present an alternative characterization of acceptable functions.

The structure of this thesis is as follows. In chapter 2, we provide some background by recalling the related definitions. In chapter 3, we start by suggesting minor modifications to the definition of elementary functions and prove that these slightly modified elementary functions are **WhileCC**-approximable (Definition 2.2.14). Then, in chapter 4 we present effective open exhaustions for the domains of elementary functions and in chapter 5, we prove that elementary functions are continuous with respect to the specific open exhaustions we presented inductively for their domains back in chapter 4. This concludes our proof of acceptability of elementary functions and implies that elementary functions are computable in all four models of computation mentioned above. In chapter 6, we conclude the thesis and discuss potential directions for future research. Then, in Appendix A, we provide some additional lemmas that are used in chapter 3. Also, the Index at the end of this thesis provides an alphabetical listing of key terms and topics discussed, allowing for quick reference.

Chapter 2

Preliminaries

In this section, we review the basic concepts of signatures and many-sorted algebras defined by Tucker and Zucker [2001] along with the **WhileCC** programming language defined by Tucker and Zucker [2004] as well as some basic concepts of real analysis.

Notation. In this thesis, we use $\rightharpoonup$ to denote partial functions and $\twoheadrightarrow$ for total functions. Contrary to the normal convention, the function log denotes $\log_2$ and ln is used for $\log_e$. We write $\bar{x}$ to denote a sequence $x_1, x_2, \ldots, x_n$ and define $|\bar{x}| = n$. Throughout this work, the equality symbol $=$ denotes Kleene equality, meaning either both sides of the equality are defined and are equal, or both sides are undefined. The symbol ϕ denotes an enumeration of all computable functions on $\mathbb{N}$ and is used in Sections 2.3 and 2.4. In this thesis, "continuity of a function on its domain" is used in the standard sense.

2.1 Basic Algebraic Concepts: Signatures and Algebras

Definition 2.1.1 (Many-sorted signatures, [Tucker and Zucker, 2001]). A signature Σ is a pair $\langle \mathbf{Sort}(\Sigma), \mathbf{Func}(\Sigma)\rangle$ where

- $\mathbf{Sort}(\Sigma)$ is a finite set of sorts, and
- $\mathbf{Func}(\Sigma)$ is a finite set of (primitive or basic) function symbols, where each $F \in \mathbf{Func}(\Sigma)$ has a type $s_1 \times \cdots \times s_n \to t$ where the arity of F is n and $s_1, s_2, \ldots, s_n, t \in \mathbf{Sort}(\Sigma)$. We write $F : s_1 \times \cdots \times s_n \to t$. The case $n = 0$ corresponds to constant symbols and can be written as $F : \to t$ or $F : t$.

Definition 2.1.2 (Σ-algebras, [Tucker and Zucker, 2001]). A partial Σ-algebra A has

- a non-empty set A_s, the *carrier of sort* s, for each $s \in \mathbf{Sort}(\Sigma)$, and
- a (possibly partial) function $F^A : A_{s_1} \times \cdots \times A_{s_m} \rightharpoonup A_t$, the *interpretation of* F, for each basic function symbol $F : s_1 \times \cdots \times s_m \to t$ in $\mathbf{Func}(\Sigma)$.

Definition 2.1.3 (Topological partial algebras, [Tucker and Zucker, 2001]). A *topological partial algebra* is a partial Σ-algebra with topologies on the carriers such that each basic function symbol's interpretation is continuous on its domain, and the carriers $\mathbb{B}$ and $\mathbb{N}$ (if present) have discrete topology.

Definition 2.1.4 (Signature of booleans: $\Sigma(\mathcal{B})$, [Tucker and Zucker, 2001]). The signature $\Sigma(\mathcal{B})$ below is the signature of *booleans*:

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

Definition 2.1.5 (Standard signature, [Tucker and Zucker, 2001]). A signature Σ is a *standard signature*, if:

- $\Sigma(\mathcal{B}) \subseteq \Sigma$, and
- for all $s \in \mathbf{Sort}(\Sigma) \setminus \Sigma(\mathcal{B})$, the function symbols of Σ include a *discriminator* $\mathsf{if}_s : \mathsf{bool} \times s \times s \to s$.
- if the function symbols of Σ include eq_s, then it has to have type $s \times s \to \mathsf{bool}$.

Definition 2.1.6 (Standard algebra, [Tucker and Zucker, 2001]). Given a standard signature Σ, a Σ-algebra A is a *standard algebra*, if:

- A has the carrier $\mathbb{B} = \{\mathsf{tt}, \mathsf{ff}\}$ for sort bool,
- A has the standard interpretations of the function and constant symbols of $\Sigma(\mathcal{B})$. Thus, for example, $\mathsf{true}^A = \mathsf{tt}$ and $\mathsf{false}^A = \mathsf{ff}$, and
- the discriminators and equality operators have their standard interpretation in A; i.e., for $b \in \mathbb{B}$ and $x, y \in A_s$

  $$\mathsf{if}_\mathsf{s}(b, x, y) = \begin{cases} x & \text{if } b = \mathsf{tt} \\ y & \text{if } b = \mathsf{ff}, \end{cases}$$

  and
- for each sort s for which the function symbol eq_s is present, it is interpeted as the identity relation on s.

Definition 2.1.7 (The topological partial algebra $\mathcal{R}$). The following algebra $\mathcal{R}$ is a topological, partial, and standard algebra that we will be working with, in this document:

algebra	$\mathcal{R}$
carriers	$\mathbb{R}$, $\mathbb{N}$, $\mathbb{B}$
functions	$0_\mathsf{real},\ 1_\mathsf{real},\ -1_\mathsf{real} : \ \twoheadrightarrow \mathbb{R}$
	$+_\mathsf{real},\ \times_\mathsf{real} : \mathbb{R} \times \mathbb{R} \twoheadrightarrow \mathbb{R}$
	$+_\mathsf{nat},\ \times_\mathsf{nat} : \mathbb{N} \times \mathbb{N} \twoheadrightarrow \mathbb{N}$
	$\mathsf{inv}_\mathsf{real} : \mathbb{R} \to \mathbb{R}$
	$0_\mathsf{nat} : \ \twoheadrightarrow \mathbb{N}$
	$\mathsf{suc}_\mathsf{nat} : \mathbb{N} \twoheadrightarrow \mathbb{N}$
	$\mathsf{tt},\ \mathsf{ff} : \ \twoheadrightarrow \mathbb{B}$
	$\mathsf{and},\ \mathsf{or} : \mathbb{B} \times \mathbb{B} \twoheadrightarrow \mathbb{B}$
	$\mathsf{not} : \mathbb{B} \twoheadrightarrow \mathbb{B}$
	$=_\mathsf{nat},\ <_\mathsf{nat} : \mathbb{N} \times \mathbb{N} \twoheadrightarrow \mathbb{B}$
	$=_\mathsf{real},\ <_\mathsf{real} : \mathbb{R} \times \mathbb{R} \to \mathbb{B}$

The signature $\Sigma(\mathcal{R})$ with sorts $\mathsf{real}, \mathsf{bool}, \mathsf{nat}$ can be inferred from this. The functions commonly known as inverse, equality and order are not continuous. Since the concepts of **While**-computability and **WhileCC**-computability We are using here have been designed to make all expressed functions continuous on their domains [Tucker and Zucker, 2001, 2004], the discontinuities in conventional models of equality, order, and inverse have been resolved by making the functions in $\mathcal{R}$ undefined there:

$$\mathsf{inv}_{\mathsf{real}}(x) = \begin{cases} 1/x & \text{if } x \neq 0 \\ \uparrow & \text{otherwise} \end{cases}$$

$$=_{\mathsf{real}}(x, y) = \begin{cases} \mathsf{ff} & \text{if } x \neq y \\ \uparrow & \text{otherwise} \end{cases}$$

$$<_{\mathsf{real}}(x, y) = \begin{cases} \mathsf{tt} & \text{if } x < y \\ \mathsf{ff} & \text{if } x > y \\ \uparrow & \text{if } x = y \end{cases}$$

2.2 Computability on $\mathbb{R}$: WhileCC Programming Language

Let us review the the **WhileCC**(Σ) programming language over an N-standard signature Σ. We will be working on the model based on the **WhileCC** programming language over $\mathcal{R}$. A prominent feature of this language is the choose operator which selects a natural number satisfying a computable predicate. As Tucker and Zucker [2004] show, any abstract model for computing functions on topological algebras, needs to have partial operations and computable functions that are continuous and multi-valued even to be able to compute deterministic problems. Hence, having a nondeterministic countable choice operation choose here is a feature and not a bug!

2.2.1 Syntax

Definition 2.2.1 (WhileCC(Σ) programming language syntax, [Tucker and Zucker, 2004]). For any N-standard algebra Σ, we define four syntactic classes:

(a) **Σ-variables** (written $\mathbf{Var}(\mathbf{\Sigma})$):
The class of $\mathbf{Var}(\mathbf{\Sigma})$ consists of variables of each sort $s \in \mathbf{Sort}(\mathbf{\Sigma})$. For each $s \in \mathbf{Sort}(\mathbf{\Sigma})$, $\mathbf{Var_s}$ denotes the class of variables of sort s: $\mathbf{Var}(\mathbf{\Sigma}) = \cup_{s \in \mathbf{Sort}(\mathbf{\Sigma})} \mathbf{Var_s}$.

(b) **Σ-terms** (written $\mathbf{Term}(\mathbf{\Sigma})$):
For each $s \in \mathbf{Sort}(\mathbf{\Sigma})$, terms are generated by:

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

where F is a Σ-function of type $s_1 \times \cdots \times s_m \to s$ (written $F : s_1 \times \cdots \times s_m \to s$) and $s_1, \ldots, s_m, s \in \mathbf{Sort}(\mathbf{\Sigma})$

(c) **Σ-statements** (written $\mathbf{Stmt}(\mathbf{\Sigma})$): Statements are generated by:

$$\begin{aligned} S ::= \mathsf{skip} \ &\mid \mathsf{div} \mid \bar{x} := \bar{t} \mid S_1 \ S_2 \mid \mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi} \\ &\mid \mathsf{while}\ b\ \mathsf{do}\ S_0\ \mathsf{od} \mid n := \mathsf{choose}\ (z : \mathsf{nat}) : P(z, \bar{t}) \end{aligned}$$

where b is of type bool and $\bar{x} := \bar{t}$ denotes valid concurrent assignment i.e., $\bar{x}$ is a tuple of *distinct* variables and $\bar{t}$ is a tuple of **Σ-terms**. Also, n and z are of type nat, P is a procedure of type $\mathsf{nat} \times \bar{w} \to \mathsf{bool}$, and $\bar{t}$ is of type $\bar{w}$.

(d) **Σ-procedures** (written **Proc**(**Σ**)):
Procedures of type $\bar{u} \to \bar{v}$ are generated by:

$$P ::= \mathsf{proc}\ D\ \mathsf{begin}\ S\ \mathsf{end}$$

where S is the body (all the variables in the body S should be defined in D), and D is the variable declaration of the form:

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

with $\bar{a}, \bar{b}$ and $\bar{c}$ being tuples of input, output, and auxiliary (distinct and pairwise disjoint) variables respectively of type $\bar{u}$, $\bar{v}$, and $\bar{w}$. Note that the aux clause can be omitted when $|\bar{c}| = 0$.

Remark 2.2.2. In this thesis, we exclusively work with **WhileCC**($\mathcal{R}$) programming language. So whenever **WhileCC** is mentioned without specifying a standard algebra, we mean **WhileCC**($\mathcal{R}$).

Remark 2.2.3. In the current setup of the language, we do not introduce names for procedures inside the grammar. Instead, we define shorthands at the metalanguage level for procedures explicitly (using the ≡ operator) or implicitly ("Let procedure P be as follows, etc."). These shorthands allow us to make the syntax more readable without complicating the grammar. For example, instead of writing

```
proc
  in $n$ : nat
  out $r$ : bool
  aux $k$ : nat
begin
  $r$ := choose ($k$ : nat) : proc
                                in $m$ : nat $n$ : nat
                                out $res$ : bool
                              begin
                                if $n =_{\mathsf{nat}} 2 \times_{\mathsf{nat}} m$ then
                                  $res$ := tt
                                else
                                  $res$ := ff
                                fi
                              end ($k, n$)
end
```

we can define a shorthand for the inner procedure

```
ExampleProc ≡ proc
                in $m$ : nat $n$ : nat
                out $res$ : bool
              begin
                if $n =_{\mathsf{nat}} 2 \times_{\mathsf{nat}} m$ then
                  $res$ := tt
                else
                  $res$ := ff
                fi
              end
```

and then the main procedure could be changed into the following:

```
proc
   in $n$ : nat
   out $r$ : bool
   aux $k$ : nat
begin
   $r$ := choose ($k$ : nat) : ExampleProc($k, n$)
end
```

Notation. For expressions $X_1, \ldots, X_n$, we write $X_1 < X_2 < \cdots < X_n$ as a shorthand for $X_1 < X_2$ and $X_2 < X_3$ and $\cdots$ and $X_{n-1} < X_n$. We use $1/x$ as a shorthand for $\mathsf{inv}_{\mathsf{real}}(x)$, x/y for $x \times_{\mathsf{real}} (1/y)$, $-x$ for $(-1_{\mathsf{real}} \times_{\mathsf{real}} x)$, and $x - y$ for $x +_{\mathsf{real}} (-y)$. We omit the subscripts on $+_{\mathsf{nat}}$ and $+_{\mathsf{real}}$ (resp. $\times_{\mathsf{nat}}$ and $\times_{\mathsf{real}}$) since the appropriate subscript can be inferred from the argument types. Also, we write if b then S_1 fi as a shorthand for if b then S_1 else skip fi. We write $n :=$ choose $(z : \mathsf{nat}) : t$ as a shorthand for

$$n := \mathsf{choose}\ (z : \mathsf{nat}) : \mathsf{ProcP}(z, \ldots)$$

where t is a Σ-term and z may appear free in t and ProcP is:

```
proc
   in $z : s \ldots$
   out $r$ : bool
begin
   $r := t$
end
```

Procedure Call Statements. Note that our definition of **WhileCC**-syntax does not involve any procedure calls other than within a choose statement. We extend our rules of statement generation to include procedure calls of the form

$$\bar{u} := P(\bar{t})$$

where P is a procedure of type $\bar{s} \to r$ with $\bar{t}$ a tuple of terms of type $\bar{s}$ and u is a variable of type r. A procedure call is syntactic sugar for copying the body of the called procedure P into the calling procedure with any necessary variable renaming, initializing P' s input variables with the term(s) $\bar{t}$ and copying the output(s) into variable(s) $\bar{u}$. Therefore, there are no recursive procedure calls. Note that procedure calls do *not* computationally strengthen our model. In the following sections, any appearance of procedure P in an expression

$$\bar{x} := \cdots P(\bar{t}) \cdots$$

is a short form of

$$\bar{y} := P(\bar{t})$$
$$\bar{x} := \cdots \bar{y} \cdots$$

for some fresh tuple y of the same type as the output type of P.

Casting nat to real. We present a **WhileCC**-program that maps each natural number to its equivalent real number below:

```
toReal ≡ proc
            in $n$ : nat
            aux $counter$ : nat $sum$ : real
         begin
            $sum := 0_{\mathsf{real}}$
            $counter := 0_{\mathsf{nat}}$
            while $counter <_{\mathsf{nat}} n$ do
               $sum := sum + 1_{\mathsf{real}}$
               $counter := counter + 1_{\mathsf{nat}}$
            od
            return $sum$
         end
```

From here on, we implicitly make use of this mapping whenever we input anything of type nat as a real argument (namely assigning nat values to real variables).

Choosing Multiple nat Values. In this thesis, we define the shorthand

$$\bar{x} := \mathsf{choose}\ (z_1 : \mathsf{nat}, \dots, z_n : \mathsf{nat}) : P(\bar{z}, \bar{t})$$

for choose for multiple distinct variables where procedure P is of type

$$\overbrace{\mathsf{nat} \times \cdots \times \mathsf{nat}}^{n \text{ times}} \times \bar{s} \to \mathsf{bool},$$

and $\bar{t}$ is a tuple of terms of type $\bar{s}$. This shorthand stands for

```
$x_1$ := choose $(z_1 : nat)$ : proc
                                 in $z_1$ : nat $\bar{r} : \bar{s}$
                                 out $res$ : bool
                                 aux $tmp_2$ : nat ... $tmp_n$ : nat
                              begin
                                 $tmp_2, \dots, tmp_n :=$
                                    choose $(z_2 : \mathsf{nat}, \dots, z_n : \mathsf{nat}) : P(\bar{z}, \bar{r})$
                                 $res :=$ tt
                              end $(z_1, \bar{t})$
$x_2, \dots, x_n :=$ choose $(z_2 : \mathsf{nat}, \dots, z_n : \mathsf{nat}) : P(x_1, z_2, \dots, z_n, \bar{t})$
```

Note that this notation is defined recursively, i.e., choose with n variables of type nat is defined in terms of choose with $n-1$ variables. This definition is by no means efficient: in fact this recursive definition leads to an exponential number of calls to choose. For n variables, choose with one variable is executed $2^n - 1$ times rather than just once. The reason is that the procedure P needs all the arguments to be chosen simultaneously.

An alternative approach that addresses the inefficiency concern would be to encode each n-tuple as a single natural number using Gödel's numbering, choose the Gödel number of the output tuple, and then decode it to obtain each value. This would reduce the number of calls to choose to only one. While this is certainly feasible, the efficiency is not our main concern here. Our primary goal here is to establish that choose can be extended to multiple variables without adding computational power to our language. A similar idea for the extension is mentioned in Tucker and Zucker [2004] for choosing "pairs" of natural numbers using primitive recursive pairing and projection functions.

Choosing rational Values. Definition 2.2.1 does not support choosing a real value. However, as mentioned in Tucker and Zucker [2004], we can extend our programming language in a conservative manner to allow choosing a rational value.

In this thesis, we simulate choosing a real value by choosing a numerator and a denominator (both of type nat), and then using division and multiplication by -1 to construct a real value. Clearly, this construction will only give us rational values.

Let P be a procedure of type $\mathsf{real} \times s_1 \times \cdots \times s_n \to \mathsf{bool}$. We define

$$x := \mathsf{choose}\ (q : \mathsf{real}) : P(q, \bar{t})$$

as a shorthand for

$$x := \mathsf{ChooseRationalSuchThatP}(\bar{t}),$$

where the procedure ChooseRationalSuchThatP is defined below:

```
ChooseRationalSuchThatP ≡
  proc
    in $\bar{t} : \bar{s}$
    out $q : real$
    aux $n_1 : \mathsf{nat}\ n_2 : \mathsf{nat}\ sign : \mathsf{nat}$
  begin
  $sign, n_1, n_2 :=$ choose $(k : \mathsf{nat}, k_1 : \mathsf{nat}, k_2 : \mathsf{nat})$ :
                     proc
                       in $k : \mathsf{nat},\ k_1 : \mathsf{nat},\ k_2 : \mathsf{nat}$
                       out $res : bool$
                       aux
                     begin
                       if $k =_{\mathsf{nat}} 0$ then
                         $res := P(k_1/(k_2+1), \bar{t})$
                       else
                         $res := P(-k_1/(k_2+1), \bar{t})$
                       fi
                     end $(k, k_1, k_2)$
  if $sign =_{\mathsf{nat}} 0$ then
    $q := n_1/(n_2+1)$
  else
    $q := -n_1/(n_2+1)$
  fi
  end
```

We extend our choose syntax to support multiple real variables in the same way we extend our syntax to support multiple nat variables.

The return Statement. We alternatively write programs with one output variable, omitting the out clause and writing a return statement at the last line. The procedure

```
proc
  in ...
  aux ...
begin
  ...
  return $t$
end
```

is a shorthand for the following:

```
proc
  in ...
  out $r : s$
  aux ...
begin
  ...
  $r := t$
end
```

Note that the type of the output variable is inferred from the type of t.

2.2.2 Semantics

In this section, we briefly review the **WhileCC**(Σ) programming language semantics originally given in Tucker and Zucker [2004]. It's worth mentioning that **WhileCC**-statements are interpreted as countably-many-valued state transformations, and procedures are interpreted as countably-many-valued functions on $\mathcal{R}$.

We begin by formally defining the concept of a state, then we define semantics for terms, and then for statements. We then conclude this section by giving the semantics of procedures.

Let us review the definition of a state:

Definition 2.2.4 (States of an algebra, [Tucker and Zucker, 2004]). Let A be a standard Σ-algebra. Then a *state* on A is a family $\langle \sigma_s \mid s \in \mathbf{Sort}(\mathbf{\Sigma}) \rangle$ of functions $\sigma_s : \mathbf{Var_s} \to A_s$ that maps each variable of sort s to an element of A_s. The set of states on A is called $\mathbf{State}(A)$.

Notation For any set A, A^* is the set of all finite sequences in A, and $\mathcal{P}^+_\omega(A)$ is the set of all countable non-empty subsets of A. Also, let $\uparrow$ denote divergence and $\downarrow$ be an infix binary symbol denoting convergence to the second argument. For a variable v of type s, we write $\sigma(v)$ for $\sigma_s(v)$ since the type of v is unambiguous. For a tuple $\bar{x}$ with $|\bar{x}| = m$, we write $\sigma[\bar{x}]$ for the tuple $(\sigma(x_1), \dots, \sigma(x_m))$. Let σ be a state on A, $\bar{x} : \bar{u}$ with $|\bar{x}| = m$, and $\bar{a} \in A_{u_1} \times \cdots \times A_{u_m}$. The *variant* $\sigma\{\bar{x}/\bar{a}\}$ is defined as

$$\sigma\{\bar{x}/\bar{a}\}(y) = \begin{cases} a_i & \text{if } y \equiv x_i \\ \sigma(y) & \text{otherwise.} \end{cases}$$

Definition 2.2.5 (Semantics of terms, [Tucker and Zucker, 2004]). The meaning of a term $t \in \mathbf{Term}_s$ in a standard algebra A is a function

$$[\![t]\!]^A : \mathbf{State}(A) \to \mathcal{P}^+_\omega(A_s \cup \{\uparrow\}).$$

The definition is by structural induction on terms:

$$\begin{aligned}
[\![x]\!]^A\sigma &= \{\sigma(x)\}\\
[\![F(t_1,\dots,t_m)]\!]^A\sigma &= \big\{y \mid \exists x_1 \in A \cap [\![t_1]\!]^A\sigma, \dots, x_m \in A \cap [\![t_m]\!]^A\sigma : F(x_1,\dots,x_m) \downarrow y\big\}\\
&\cup \big\{\uparrow \mid \exists x_1 \in A \cap [\![t_1]\!]^A\sigma, \dots, x_m \in A \cap [\![t_m]\!]^A\sigma : F(x_1,\dots,x_m)\uparrow\big\}\\
&\cup \big\{\uparrow \mid \exists\, 1 \le i \le m \quad \uparrow \in [\![t_i]\!]^A\sigma\big\}\\
[\![\mathsf{if}(b,t_1,t_2)]\!]^A\sigma &= \big\{y \mid (\mathsf{tt} \in [\![b]\!]^A\sigma \wedge y \in [\![t_1]\!]^A\sigma) \;\vee\; (\mathsf{ff} \in [\![b]\!]^A\sigma \wedge y \in [\![t_2]\!]^A\sigma)\big\}\\
&\cup \big\{\uparrow \mid \uparrow \in [\![b]\!]^A\sigma\big\}
\end{aligned}$$

In order to define the semantics for statements, the algebraic operational method described in Tucker and Zucker [2001] is used. Using this method, in order to define statements as state transformations, one should provide two main components, which will be defined below in Definition 2.2.7:

- a strict subset of statements, *atomic statements* written **AtSt**, along with a meaning function $(\!|\ \_\ |\!)^A : \mathbf{AtSt} \to \mathbf{State}(A) \to \mathbf{State}(A)$, and
- functions

  $$\mathbf{First} : \mathbf{Stmt} \to \mathbf{AtSt}$$

  and

  $$\mathbf{Rest} : \mathbf{Stmt} \times \mathbf{State(A)} \to \mathcal{P}^+_\omega(\mathbf{Stmt}) :$$

  First(S) intuitively gives the first step in execution of S in any state, and **Rest**(S,σ) is a finite set of statements that gives the rest of the execution of statement S in state σ. The result of **Rest** depends on the start state for if-then-else statements and while loops, where the condition needs to be evaluated in that state to determine the remaining statements to execute. In these cases, **First** produces the skip statement, and can therefore be deterministic.

Definition 2.2.6 (Semantics of atomic statements, [Tucker and Zucker, 2004]). A statement is *atomic* if it is of the form skip, $x := t$, or $x := \mathsf{choose}(z : nat) : P(z, \bar{t})$. The set of all atomic statements is denoted by **AtSt**. The meaning of atomic statements in a standard algebra A is a function

$$(\!|\ \ |\!)^A : \mathbf{AtSt} \to \mathbf{State}(A) \to \mathcal{P}^+_\omega(\mathbf{State}(A) \cup \{\uparrow\})$$

defined by

$$\begin{aligned}
(\!|\ \mathsf{div}\ |\!)^A\sigma &= \{\uparrow\}\\
(\!|\ \mathsf{skip}\ |\!)^A\sigma &= \{\sigma\}\\
(\!|\ x := t\ |\!)^A\sigma &= \big\{\sigma\{x/a\} \mid a \in A \cap [\![t]\!]^A\sigma\big\}\\
&\cup \big\{\uparrow \mid \uparrow \in [\![t]\!]^A\sigma\big\}\\
(\!|\ x := \mathsf{choose}(z : nat) : P(z,\bar{t})\ |\!)^A\sigma &= \big\{\sigma\{x/n\} \mid n \in \mathbb{N} \wedge \mathsf{tt} \in [\![P]\!]^A(n, \sigma[\bar{t}])\big\}\\
&\cup \big\{\uparrow \mid \forall n \in \mathbb{N}(\mathsf{tt} \notin [\![P]\!]^A(n, \sigma[\bar{t}]))\big\}
\end{aligned}$$

Now we need to provide the functions **First** and **Rest** for each of the non-atomic statements.

Definition 2.2.7 (The functions First and Rest, [Tucker and Zucker, 2004]). Let A be a standard algebra, $S \in \mathbf{Stmt}(\Sigma)$, and $\sigma \in \mathbf{State}(A)$. Then the function

$$\mathbf{First} : \mathbf{Stmt} \to \mathbf{AtSt}$$

is defined as

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

The function

$$\mathbf{Rest}^A : \mathbf{Stmt} \times \mathbf{State}(A) \to \mathcal{P}^+_\omega(\mathbf{Stmt})$$

is defined as follows:

- If S is atomic, then $\mathbf{Rest}^A(S, \sigma) = \{\mathsf{skip}\}$.
- If $S \equiv S_1 S_2$ and S_1 is atomic, then $\mathbf{Rest}^A(S, \sigma) = \{S_2\}$. If $S \equiv S_1 S_2$ and S_1 is not atomic, then $\mathbf{Rest}^A(S, \sigma) = \{S' S_2 \mid S' \in \mathbf{Rest}(S_1, \sigma)\} \cup \big\{\mathsf{div} \mid \mathsf{div} \in \mathbf{Rest}^A(S_1, \sigma)\big\}$.
- If $S \equiv \mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}$, then
$$\mathbf{Rest}^A(S, \sigma) = \big\{S_1 \mid \mathsf{tt} \in [\![b]\!]^A \sigma\big\} \cup \big\{S_2 \mid \mathsf{ff} \in [\![b]\!]^A \sigma\big\} \cup \big\{\mathsf{div} \mid \uparrow \in [\![b]\!]^A \sigma\big\}$$
- If $S \equiv \mathsf{while}\ b\ \mathsf{do}\ S_0\ \mathsf{od}$, then
$$\mathbf{Rest}^A(S, \sigma) = \big\{S_0 S \mid \mathsf{tt} \in [\![b]\!]^A \sigma\big\} \cup \big\{\mathsf{skip} \mid \mathsf{ff} \in [\![b]\!]^A \sigma\big\} \cup \big\{\mathsf{div} \mid \uparrow \in [\![b]\!]^A \sigma\big\}$$

The above three components induce a semantics for the statements $[\![\ ]\!]^A : \mathbf{Stmt} \to \mathbf{State}(A) \to \mathcal{P}^+_\omega(\mathbf{State}(A) \cup \{\uparrow\})$.

In order to define the semantics of statements, we need to define the concepts "computation step", "computation tree stage" and "computation tree".

Definition 2.2.8 (Computation step function, [Tucker and Zucker, 2004]). Let A be a standard algebra. We define the *computation step* function

$$\mathbf{CompStep}^{\mathbf{A}} : \mathbf{Stmt} \times \mathbf{State}(A) \twoheadrightarrow \mathcal{P}^+_\omega(A \cup \{\uparrow\})$$

by

$$\mathbf{CompStep}^{\mathbf{A}}(S, \sigma) = \llparenthesis\ \mathbf{First}(S)\ \rrparenthesis^A \sigma$$

Definition 2.2.9 (Computation tree stage, [Tucker and Zucker, 2004]). Let A be a standard algebra. We define a *computation tree stage* function

$$\mathbf{CompTreeStage}^{\mathbf{A}} : \mathbf{Stmt} \times \mathbf{State}(A) \times \mathbb{N} \twoheadrightarrow \mathcal{P}^+_\omega(\mathbf{State}(A \cup \{\uparrow\})^*)$$

by induction

- Induction base: $\mathbf{CompTreeStage}^{\mathbf{A}}(S, \sigma, 0) = \{\sigma\}$

- Induction step: $\mathbf{CompTreeStage^A}(S, \sigma, n)$ is formed by attaching to the root $\{\sigma\}$ the following:
  - if S is atomic: the leaf $\{\sigma'\}$ for each $\sigma' \in (\!|\ S\ |\!)^A\sigma$;
  - if S is not atomic: the subtree $\mathbf{CompTreeStage^A}(S', \sigma', n-1)$, for each $\sigma' \in \mathbf{CompStep^A}(S, \sigma)$ with $\sigma \neq\ \uparrow$ and $S' \in \mathbf{Rest}^A(S, \sigma)$, as well as the leaf $\{\uparrow\}$ if $\uparrow \in \mathbf{CompStep^A}(S, \sigma)$

Definition 2.2.10 (Computation tree, [Tucker and Zucker, 2004]). Let A be a standard algebra. Then we define the function $\mathbf{CompTree^A}$ by

$$\mathbf{CompTree^A}(S, \sigma) \overset{def}{=} \lim_{n\to\infty} \mathbf{CompTreeStage^A}(S, \sigma, n).$$

Remark 2.2.11. Definition 2.2.10 defines an ω-branching tree, branching according to possible output states. Each node of this tree is labeled by either a state, or "$\uparrow$". Furthermore, "$\uparrow$" can only be a leaf node of this tree.

Note that the limit above is well-defined, since each computation stage tree

$$\mathbf{CompTreeStage^A}(S, \sigma, n+1)$$

is an extension of

$$\mathbf{CompTreeStage^A}(S, \sigma, n),$$

where exactly one layer of leaves is attached to leaves in $\mathbf{CompTreeStage^A}(S, \sigma, n)$.

Any actual computation of a statement S at state σ corresponds to one of the paths in this tree.

Definition 2.2.12 (Semantics of WhileCC-statements, [Tucker and Zucker, 2004]). Let A be a standard algebra. Then we define the semantics function

$$[\![\ ]\!]^A : \mathbf{Stmt} \to \mathbf{State}(A) \to \mathcal{P}^+_\omega(\mathbf{State}(A \cup \{\uparrow\}))$$

as

$$[\![S]\!]^A\sigma \overset{def}{=} \{\text{leaves in } \mathbf{CompTree^A}(S, \sigma)\} \cup \{\uparrow \mid \mathbf{CompTree^A}(S, \sigma) \text{ has an infinite path}\}.$$

Finally we can define the semantics for **WhileCC**-procedures:

Definition 2.2.13 (Semantics of WhileCC-procedures, [Tucker and Zucker, 2004]). The semantics of a **WhileCC**-procedure P of type $\bar{u} \to \bar{v}$ with the definition

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

is a (many-valued) function:

$$P^{\mathcal{R}} : \mathcal{R}_{\bar{u}} \to \mathcal{P}^+_\omega(\mathcal{R}_{\bar{v}} \cup \{\uparrow\})$$

defined as

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

Now let us review the concept of **WhileCC**-approximability defined in Fu and Zucker [2014]. Let $P : \mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure. Then $P^{\mathcal{R}}_n$ is defined by $P^{\mathcal{R}}_n \overset{def}{=} P^{\mathcal{R}}(\cdot, n) : \mathbb{R} \to \mathcal{P}^+_\omega(\mathbb{R} \cup \{\uparrow\})$.

Definition 2.2.14 (WhileCC-approximability, [Fu and Zucker, 2014]). A **WhileCC**-procedure P of type $\mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ on $\mathcal{R}$ is said to *approximate* a function $f : \mathbb{R} \to \mathbb{R}$ iff for all $n \in \mathbb{N}$ and all $x \in \mathbb{R}$:

(i) $x \in \mathbf{dom}(f) \implies \uparrow \notin P_n^{\mathcal{R}}(x)$, and

(ii) $x \in \mathbf{dom}(f) \implies P_n^{\mathcal{R}}(x) \subseteq \mathbf{Nbd}(f(x), 2^{-n})$, and

(iii) $x \notin \mathbf{dom}(f) \implies P_n^{\mathcal{R}}(x) = \{\uparrow\}$

where $\mathbf{Nbd}(y, r)$ has the standard definition of neighborhood on $\mathbb{R}$ i.e.,

$$\mathbf{Nbd}(y, r) = \{z \in \mathbb{R} \ \mid |y - z| < r\}.$$

Definition 2.2.15 (Well-defined WhileCC-procedures). We call a procedure $P : \bar{u} \to \bar{v}$ *well-defined* iff for $\bar{x} : \bar{u}$ we have

$$P^{\mathcal{R}}(\bar{x}) = \{\uparrow\}, \text{ or } \uparrow \notin P^{\mathcal{R}}(\bar{x}).$$

In this thesis, since all **WhileCC**-procedures of interest are well-defined [1], we consider the semantics of a procedure $P : \bar{u} \to \bar{v}$ to be a function

$$P^{\mathcal{R}} : \mathcal{R}_{\bar{u}} \to \mathcal{P}_{\omega}(\mathcal{R}_{\bar{v}})$$

where $P^{\mathcal{R}}(\bar{x}) = \emptyset$ iff P does not terminate on input $\bar{x}$.

Definition 2.2.16 (WhileCC-computability on $\mathbb{N}$). A function $f : \mathbb{N}^k \to \mathbb{N}$ is **WhileCC**-computable if there is a **WhileCC** procedure P such that $f = P^{\mathcal{R}}$.

Remark 2.2.17. The classical Böhm and Jacopini [1966] theorem states that Turing Machines can be simulated in any programming language with composition and iteration. This, along with the Church-Turing thesis, implies that **WhileCC**-computability of a function of type $\mathbb{N}^k \to \mathbb{N}$ is equivalent to computability via any effective method. This means that from here on, in order to show that a (total) function of type $\mathbb{N}^k \to \mathbb{N}$ is recursive, it suffices to give a (necessarily terminating) algorithm for computing it [Cutland, 1980]. Note that the terms "recursive", "computable", and "**WhileCC**-computable" could be used interchangeably for functions of type $\mathbb{N}^k \to \mathbb{N}$. In other words, we do not care which model of computation is used to compute functions of this type. As long as a function is computable in some model, it is computable in every model.

2.2.3 Some Basic Functions

Using $\times_{\mathsf{real}}$ we can define the procedure $\mathsf{pow} : \mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ as:

[1] This can be easily proven, although we do not provide a proof for it.

```
pow ≡ proc
    in $a$ : real $n$ : nat
    aux $counter$ : nat $r$ : real
  begin
    $r := 1_{\mathsf{real}}$
    $counter := 0_{\mathsf{nat}}$
    while $counter <_{\mathsf{nat}} n$ do
      $r := r \times a$
      $counter := counter + 1$
    od
    return $r$
  end
```

This lets us use x^n as a shorthand for $\mathsf{pow}(x, n)$ and x^{-n} for $\mathsf{inv}_{\mathsf{real}}(\mathsf{pow}(x, n))$ with x being of type real and n of type nat.
We can also define the **WhileCC**-procedure Factorial : nat → nat as below:

```
Factorial ≡ proc
    in $n$ : nat
    aux $counter$ : nat $res$ : nat
  begin
    $res := 1_{\mathsf{nat}}$
    $counter := 1_{\mathsf{nat}}$
    while $counter <_{\mathsf{nat}} n + 1$ do
      $res := res \times counter$
      $counter := counter + 1$
    od
    return $res$
  end
```

Proposition 2.2.18. We can add the operator abs defined as

$$\mathsf{abs}(x) \simeq \begin{cases} x & \text{if } x > 0 \\ -x & \text{if } x < 0 \\ \uparrow & \text{if } x = 0 \end{cases}$$

to our algebra without strengthening it since it can easily be computed using:

```
abs ≡ proc
    in $x$ : real
    aux $r$ : real
  begin
    if $0 <_{\mathsf{real}} x$ then
      $r := x$
    else
      $r := -x$
    fi
    return $r$
  end
```

2.3 Computability on other domains: Realizability theory

In this section, we formally define computability of functions over sets other than $\mathbb{N}$. This is used implicitly throughout this paper. These definitions are based on Bauer [2022].

Definition 2.3.1 (Realizability relation). A relation $\Vdash_X \subseteq \mathbb{N} \times X$ is called a *realizability relation on* X, if $\Vdash_X$ is surjective and univalent. For any $x \in X$, and $n \in \mathbb{N}$, We say n *realizes*, or *represents* x, if $n \Vdash_X x$.

Definition 2.3.2 (Assembly). We call the pair $(X, \Vdash_X)$ an *assembly* if $\Vdash_X$ is a realizability relation on X.

Definition 2.3.3 (Computability on assemblies). Let $(A, \Vdash_A), (B, \Vdash_B)$ be assemblies. Then A function $f : A \to B$ is *computable* (or *recursive*) *with respect to* $\Vdash_A$ *and* $\Vdash_B$, if there is a recursive function $f' : \mathbb{N} \to \mathbb{N}$ with:

$$\forall a \in \mathbf{dom}(f) \quad \forall n \in \mathbb{N} \quad n \Vdash_A a \implies f'(n) \Vdash_B f(a)$$

The following realizability relations will be assumed by default when discussing computability on rationals, pairs and finite sequences without mentioning the respective realizability relations:

Definition 2.3.4 (Realizability relation on $\mathbb{N}$). We define the realizability relation $\Vdash_\mathbb{N}$ as the smallest relation satisfying $n \Vdash_\mathbb{N} n$ for all $n \in \mathbb{N}$.

Definition 2.3.5 (Realizability relation on $\mathbb{Q}$). Let us consider Godel's pairing function $g(x/y) = 2^x(2y + 1) - 1$. We define the realizability relation $\Vdash_\mathbb{Q}$ as:

$$n \Vdash_\mathbb{Q} \frac{p}{q} \iff \gcd(p, q) = 1 \quad \wedge \quad n = \begin{cases} 2(g(p/q) + 1) - 1 & \text{if } p/q > 0 \\ 2g(p/q) & \text{if } p/q < 0, \end{cases}$$

Definition 2.3.6 (Realizability relation on Cartesian product). Let $(A, \Vdash_A)$ and $(B, \Vdash_B)$ be assemblies. Then we define the realizability relation $\Vdash_{A\times B}$ as the smallest relation satisfying

$$\frac{m_1 \Vdash_A a \qquad m_2 \Vdash_B b}{2^{m_1}3^{m_2} \Vdash_{A\times B} (a, b)} .$$

Definition 2.3.7 (Realizability relation on sequences). Let $(A, \Vdash_A)$ be an assembly and A^* represent the set of all finite sequences on A. We define the realizability relation $\Vdash_{A^*}$ as the smallest relation satisfying

$$\frac{m_1 \Vdash_A a_1 \qquad \cdots \qquad m_n \Vdash_A a_n}{p_1^{m_1} \cdots p_n^{m_n} \Vdash_{A^*} (a_1, \dots a_n)} ,$$

where p_i is the ith prime number.

Definition 2.3.8 (Realizability relation on functions). Let $(A, \Vdash_A)$ and $(B, \Vdash_B)$ be assemblies. Then we define the realizability relation $\Vdash_{A\to B}$ as the smallest relation satisfying

$$\frac{\forall n \in \mathbb{N}\ \forall x \in A\ (n \Vdash_A x \implies \phi_i(n) \Vdash_B f(x))}{i \Vdash_{A\to B} f}$$

2.4 Computability on $\mathbb{R}$: α-tracking Computability

In this section, we review the concept of α-computability, which is central to the definition of elementary functions defined in 3.1, and an equivalence lemma by Fu and Zucker [2014].

Definition 2.4.1 (Standard enumeration of $\mathbb{Q}$, [Fu and Zucker, 2014]). Let $\alpha : \mathbb{N} \to \mathbb{Q}$. We call α a *standard enumeration* for $\mathbb{Q}$ if α is bijective, and the field operations on $\mathbb{Q}$ $(+, \cdot, -, /)$ are recursive under α, i.e.,

- there is a recursive function $add : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that
$$\forall n_1, n_2 \in \mathbb{N} \quad \alpha(n_1) + \alpha(n_2) = \alpha(add(n_1, n_2)),$$
- there is a recursive function $mult : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that
$$\forall n_1, n_2 \in \mathbb{N} \quad \alpha(n_1) \cdot \alpha(n_2) = \alpha(mult(n_1, n_2)),$$
- there is a recursive function $sub : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that
$$\forall n_1, n_2 \in \mathbb{N} \quad \alpha(n_1) - \alpha(n_2) = \alpha(sub(n_1, n_2)),$$

  and
- there is a recursive function $div : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that
$$\forall n_1, n_2 \in \mathbb{N} \quad n_2 \neq 0 \implies \alpha(n_1)/\alpha(n_2) = \alpha(div(n_1, n_2))$$

Theorem 2.4.2. There is a standard enumeration for $\mathbb{Q}$.

Proof. Let us consider Gödel's pairing function $g(x/y) = 2^x(2y+1)-1$. This is a bijection between positive rationals and $\mathbb{N}$. We can tweak this to give us a bijection f between $\mathbb{Q}$ and $\mathbb{N}$. Let us define

$$f(x/y) = \begin{cases} 2(g(x/y)+1)-1 & \text{if } x/y > 0 \\ 2g(x/y) & \text{if } x/y < 0, \end{cases}$$

This maps positive rationals to odd natural numbers and negative rationals to evens. The bijection f^{-1} is a standard enumeration for $\mathbb{Q}$ since the functions $add, mult, sub, div$ are easily proven to be recursive. □

Definition 2.4.3 (Computable reals codes Ω, [Fu and Zucker, 2014]). Let $\langle -, - \rangle : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be a recursive encoding of pairs. The set $\Omega \subset \mathbb{N}$ is the set of all codes $\langle e, m \rangle$ such that

- e is an index for a total recursive function $\phi_e : \mathbb{N} \to \mathbb{N}$ generating a Cauchy sequence
$$\alpha(\phi_e(0)), \alpha(\phi_e(1)), \alpha(\phi_e(2)), \ldots$$
  of elements of $\mathbb{Q}$.
- m is the index of a computable modulus of convergence $\phi_m : \mathbb{N} \twoheadrightarrow \mathbb{N}$, ensuring:
$$\forall k, l \geq \phi_m(n), \quad |\alpha(\phi_e(k)) - \alpha(\phi_e(l))| < 2^{-n}.$$

Definition 2.4.4 **(α-(tracking) computable reals $\mathbb{R}_c$, [Fu and Zucker, 2014]).** For any standard enumeration α, we define an enumeration of computable reals $\bar{\alpha} : \Omega \to \mathbb{R}$ to be

$$\bar{\alpha}(\langle e, m\rangle) = \lim_{i\to\infty} \alpha(\phi_e(i)).$$

The range of $\bar{\alpha}$ is called *the set of α-tracking computable reals (or α-computable reals)* and is denoted with $\mathbb{R}_c$.

Definition 2.4.5 **(α-tracking function, [Fu and Zucker, 2014]).** Let α be a standard enumeration of $\mathbb{Q}$. For functions $f : \mathbb{R} \to \mathbb{R}$ and $\tau : \mathbb{N} \to \mathbb{N}$, τ is an α-tracking function for f if:

1. If $f(\bar{\alpha}(k))$ is defined, then $\tau(k)$ is defined, and:
$$f(\bar{\alpha}(k)) = \bar{\alpha}(\tau(k)).$$
2. If $f(\bar{\alpha}(k))$ is undefined, then $\tau(k)$ is undefined.

Definition 2.4.6 **(α-(tracking) computability, [Fu and Zucker, 2014]).** The function $f : \mathbb{R} \to \mathbb{R}$ is an *α-tracking computable (also called an α-computable function)* iff it has a recursive α-tracking function.

Corollary 2.4.7. [2] Let $f : \mathbb{R} \to \mathbb{R}$ be a **WhileCC**-approximable function and $\alpha : \mathbb{N} \to \mathbb{Q}$ be any standard enumeration for $\mathbb{Q}$. Then f is α-computable.

Note that corollary 2.4.7 implies that for any two standard enumerations α_1 and α_2 for $\mathbb{Q}$, the set of α_1-computable and α_2-computable functions coincide under the assumption of **WhileCC**-approximability.

Using the Definition 2.4.6, we can easily see the α-computability of the constant function(for α-computable reals) and the identity function:

Lemma 2.4.8. The constant function $f(x) = c$ for any α-computable real c is α-computable.

Proof. Using the definition of α-computability, it suffices to show that the constant function has a recursive α-tracking function.
Since c is an α-computable real, this means $c \in \mathbb{R}_c$. Since by definition, α is a surjection, $\alpha^{-1}(c) \neq \emptyset$. So let's take an arbitrary $y \in \alpha^{-1}(c)$ and let us define $\tau(x) = y$ for all x. Then τ is a recursive α-tracking function for f. $\square$

Lemma 2.4.9. The identity function $id(x) = x$ is α-computable.

Proof. Using the definition of α-computability, it suffices to show that the identity function has a recursive α-tracking function.
let τ be the identity function on $\mathbb{N}$. Then τ is a recursive α-tracking function for $id(x)$. $\square$

2.5 Basic Real Analysis Concepts: Open Exhaustion, Effective Open Exhaustions, Continuity and Acceptability

In this thesis, we call a function *acceptable* (Definition 2.5.4) if it has the properties previously defined in Tucker and Zucker [2005] and referred to as *the global assumptions* by Fu and Zucker [2014].

[2]This is a corollary to Theorem (A) in Tucker and Zucker [2004]

Effective uniform continuity (not to be confued with "effective local uniform continuity" in Definition 2.5.3) is central to the standard definition of *total* computable functions on reals given by Gryzegorczyk and Lacombe, as published in Pour-El and Richards [1989].

As mentioned in Tucker and Zucker [2005], acceptability is a natural generalization of effective uniform continuity to *partial* functions.

In this section, we review the relevant definitions needed for defining acceptability.

Notation. Throughout this work, all topological terms like openness and closure of sets on $\mathbb{R}$, are considered with respect to the standard (Euclidean) topology. The closure of a set U is denoted by $\overline{U}$.

Definition 2.5.1 (Open exhaustion, [Fu and Zucker, 2014]). Let U be an open subset of $\mathbb{R}$, and $X = (U_0, U_1, U_2, ...)$ a sequence of open subsets of $\mathbb{R}$. Then the sequence X is called an *open exhaustion of* U iff:

1. $U = \bigcup_{l=0}^{\infty} U_l$, and
2. for each $l \in \mathbb{N}$, U_l is a finite union of non-empty open finite intervals $I_1^l, I_2^l, ..., I_{k_l}^l$ whose closures are pairwise disjoint, and
3. for each $l \in \mathbb{N}$, $\overline{U_l} = \bigcup_{i=1}^{k_l} \overline{I_i^l} \subseteq U_{l+1}$.

For each l, U_l is called a *stage* of the exhaustion, with *components* $I_1^l, I_2^l, ..., I_{k_l}^l$.

Now that we have the definition of open exhaustion, we also want to be able to *compute* the intervals in each stage of an open exhaustion.

Definition 2.5.2 (Effective open exhaustion, [Fu and Zucker, 2014]). An open exhaustion $(U_1, U_2, \ldots)$ of an open set $U \subseteq \mathbb{R}$ is called an *effective open exhaustion* if

- for all l, the components I_i^l that are intervals building up the stage U_l, are *rational* and *ordered* i.e., $I_i^l = (a_i^l, b_i^l)$ for some $a_i^l, b_i^l \in \mathbb{Q}$ where $b_i^l < a_{i+1}^l$ for $i = 1, ..., k_l - 1$, and
- the map
$$l \mapsto (a_1^l, b_1^l, ..., a_{k_l}^l, b_{k_l}^l)$$
which delivers the sequence of stages $U_l = I_1^l \cup ... \cup I_{k_l}^l$ is recursive.

Definition 2.5.3 (Effective local uniform continuity, [Fu and Zucker, 2014]). A function f on U is *effectively locally uniformly continuous w.r.t. an effective exhaustion* $(U_n)_{n\in\mathbb{N}}$ of U, if there is a recursive function $M : \mathbb{N}^2 \twoheadrightarrow \mathbb{N}$ such that for all $k, l \in \mathbb{N}$ and all $x, y \in U_l$:

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

Definition 2.5.4 (Acceptable Function). A function $f : \mathbb{R} \to \mathbb{R}$ is *acceptable* if there exists a sequence X where:

(i) X is an effective open exhaustion for $\mathbf{dom}(f)$, and

(ii) f is effectively locally uniformly continuous w.r.t. X.

2.6 Computability on $\mathbb{R}$: Multipolynomial approximability

In this section, we review the concept of multipolynomial computability, which is only used in section 5.3. During this section, whenever we refer to a "polynomial", we mean a $\mathbb{Q}$-polynomial. The symbol $\restriction$ is used for domain restriction.

Definition 2.6.1 (Multipolynomial, [Fu and Zucker, 2014]). Given a finite sequence of polynomials $(p_1, p_2, \dots, p_k)$ and a sequence of open intervals $(I_1, I_2, \dots, I_k)$ with disjoint closures, we define a ($\mathbb{Q}$-)multipolynomial $q(x)$ with domain $\bigcup_{i=1}^{k} \overline{I_i}$ as follows:

$$q(x) = \begin{cases} p_1(x) & \text{if } x \in \overline{I_1} \\ p_2(x) & \text{if } x \in \overline{I_2} \\ \dots & \\ p_k(x) & \text{if } x \in \overline{I_k} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

We denote this polynomial by

$$q = [p_1 \restriction \overline{I_1}, \dots, p_k \restriction \overline{I_k}].$$

Definition 2.6.2 (Effective sequence of multipolynomials, [Fu and Zucker, 2014]). Given an effective open exhaustion $(U_n)_{n\in\mathbb{N}}$ of U, a sequence of polyomials (q_l) is called an *effective sequence of multipolynomials* if

$$q_l = \left[p_1^l \restriction \overline{I_1^l}, \dots, p_k^l \restriction \overline{I_{k_l}^l} \right]$$

where for any $l \in \mathbb{N}$, we have $U_l = I_1^l \cup \dots \cup I_{k_l}^l$ and $(p_1, p_2, \dots, p_k)$ is an effective sequence of tuples of polynomials.

Definition 2.6.3 (Effective local multipolynomial approximability, [Fu and Zucker, 2014]). Let $f : \mathbb{R} \to \mathbb{R}$ and $X = (U_1, U_2, \dots)$ be an effective open exhaustion of $\mathbf{dom}(f)$, and let $(q_n)_{n\in\mathbb{N}}$ be an effective sequence of multipolynomials. If there is a recursive function $M : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that for all $k, l, n \in \mathbb{N}$ and $x \in \overline{U_l}$ we have

$$n \geq M(k, l) \implies |q_n(x) - f(x)| < 2^{-k},$$

we say f is *effectively locally multipolynomially approximable w.r.t. X by (q_l) via M* and write $(q_l) \twoheadrightarrow_{M,X} f$.

We say f *is effectively locally multipolynomially approximable w.r.t. X by (q_l)* if f is effectively locally multipolynomially approximable w.r.t. X via some recursive function M and write $(q_l) \twoheadrightarrow_X f$.

Now, let us review the equivalence lemma proved by Fu and Zucker [2014]:

Theorem 2.6.4 (Fu and Zucker's Equivalence Theorem). For any acceptable function $f : \mathbb{R} \to \mathbb{R}$ and any effective open exhaustion X of $\mathbf{dom}(f)$, the following are equivalent:

- f is an α-computable function.
- f is **WhileCC**-approximable.

- f is GL-computable [3] w.r.t. X.
- f is effectively locally uniformly multipolynomially approximable w.r.t. X.

[3]The definition of Grzegorczyk-Lacome computability (GL-computability) for total functions is given in [Pour-El and Richards, 1989, page 25] and later extended in Fu and Zucker [2014] to *Grzegorczyk-Lacome computability (GL-computability) with respect to some open exhaustion* for partial functions.

Chapter 3

WhileCC-Approximability of Elementary Functions

In this section, we prove that the elementary functions (Definition 3.1.1) are **WhileCC**-approximable. Our proof is structured as follows:

1. We first prove that basic elementary functions (constant and identity functions) are **WhileCC**-approximable.
2. We then prove that the composition of any elementary function with a **WhileCC**-approximable function builds a **WhileCC**-approximable function.
3. Using the first two parts, we can then inductively prove that all elementary functions are approximable by **WhileCC**-programs. This gives us the main result of the current chapter in Theorem 3.12.1.

We first define the elementary functions formally in Section 3.1, then we introduce some preliminary lemmas in Subsection 3.2. In Subsection 4.2 we prove that basic elementary functions (constant and identity functions) are **WhileCC**-approximable. In subsections 3.4 through 3.11, we prove that the composition of basic elementary function constructors (addition, multiplication, division, exponential, ln, sin, nth root and arcsin) with other **WhileCC**-approximable functions builds **WhileCC**-approximable functions, and then in Section 3.12 we conclude that all elementary functions are **WhileCC**-approximable.

3.1 Elementary Functions

Elementary functions appear in different computational contexts.

Definition 3.1.1 (Elementary functions). The elementary functions on $\mathbb{R}$ [Tenenbaum and Pollard, 1985, page 17] are partial functions defined by expressions built up from

- α-computable reals (see Remark 3.1.2 and Section 2.4), and
- the variable x

and, by applying (repeatedly) the basic operations below on elementary functions f, g:

(i) addition (i.e. $(f + g)(x) = f(x) + g(x)$)

(ii) multiplication (i.e. $(f \cdot g)(x) = f(x)g(x)$)

(iii) division (i.e. $\text{div}_f(x) = \frac{1}{f(x)}$ where $\frac{1}{0}$ is undefined)

(iv) exponential (i.e. $\exp_f(x) = e^{f(x)}$)

(v) logarithm (i.e. $\ln_f(x) = \ln(f(x))$)

(vi) $\sin_f(x) = \sin(f(x))$

(vii) n-th roots: $\text{root}_{n,f}(x) = \sqrt[n]{f(x)}$ where $0 < n \in \mathbb{N}$

(viii) $\arcsin_f(x) = \arcsin(f(x))$

In this thesis, we make the following modifications to the natural definition of some of the functions above, to work with a computable part of $\mathbb{R}$, and also to make sure the domains of those functions are open:

- We define $\sqrt[n]{x} = 0$ for $x < 0$ when n is even.
- We extend the definition of $\arcsin(x)$ to be $\frac{\pi}{2}$ for $x > 1$ and to be $-\frac{\pi}{2}$ for $x < -1$.

From here on, the term *elementary functions* will be used to refer to the modified version of unary elementary functions.

Remark 3.1.2. Note that the original definition of elementary functions by [Tenenbaum and Pollard, 1985, page 17] involves expressions built up from arbitrary (possibly non-computable) constants. We cannot approximate non-computable constants, so we choose to omit them from our definition of elementary functions and only work with α-computable reals.

3.2 Preliminary Lemmas

Definition 3.2.1. Let F : real × nat → real be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$, and $x, b \in \mathbb{R}$. We define $(F(-) < -)$ of type real × real → bool as

```
(F(−) < −) ≡ proc
                 in x : real b : real
                 aux c : nat
             begin
                 c := choose (k : nat) : F(x, k) + 2^{−k} < b
                 /* If terminates it means f(x) < b */
                 return tt
             end
```

The semantics of the procedure above is

$$(F(x) < b)^{\mathcal{R}} = \{\text{tt} \mid \exists y, n \quad y \in F^{\mathcal{R}}(x, n) \wedge y + 2^{-n} < b\}.$$

Note that instead of $(F(-) < -)^{\mathcal{R}}(x, b)$, we write $(F(x) < b)^{\mathcal{R}}$.

Definition 3.2.2. Let F : real × nat → real be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$, and and $x, b \in \mathbb{R}$. We define $(F(-) > -)$ of type real × real → bool as

```
$(F(-) > -) \equiv$ proc
    in $r$ : real $b$ : real
    aux $c$ : nat
  begin
    $c :=$ choose $(k : \mathsf{nat}) : F(r,k) - 2^{-k} > b$
    /* If terminates it means $f(r) > b$ */
    return tt
  end
```

The semantics of the procedure above is

$$(F(x) > b)^{\mathcal{R}} = \{\mathsf{tt} \mid \exists y, n \quad y \in F^{\mathcal{R}}(x,n) \wedge y - 2^{-n} > b\}.$$

Note that instead of $(F(-) > -)^{\mathcal{R}}(x,b)$, we write $(F(x) > b)^{\mathcal{R}}$.

Lemma 3.2.3. Let $F : \mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$. Then we have:

$$f(x) < y \iff \exists n \in \mathbb{N} \quad \exists y' \in F^{\mathcal{R}}(x,n) \quad y' + 2^{-n} < y \iff (F(x) < y)^{\mathcal{R}} = \{\mathsf{tt}\}. \tag{3.1}$$

Similarly,

$$y < f(x) \iff \exists n \in \mathbb{N} \quad \exists y' \in F^{\mathcal{R}}(x,n) \quad y < y' - 2^{-n} \iff (y < F(x))^{\mathcal{R}} = \{\mathsf{tt}\}. \tag{3.2}$$

Proof. We begin by proving (3.1).

($\Rightarrow$) If $f(x) < y$ then $y - f(x) > 0$, and there is $n \in \mathbb{N}$ such that $2^{-n} < \frac{y-f(x)}{2}$. Then, for any $y' \in F^{\mathcal{R}}(x,n)$ we have $f(x) - 2^{-n} < y' < f(x) + 2^{-n}$ and hence:

$$\begin{aligned} y' + 2^{-n} &< f(x) + 2^{-n+1} \\ &< f(x) + 2\left(\frac{y - f(x)}{2}\right) \\ &\le f(x) + y - f(x) \\ &\le y \end{aligned}$$

($\Leftarrow$) We assume that $\exists n \in \mathbb{N}\ \exists y' \in F^{\mathcal{R}}(x,n) \quad y' + 2^{-n} < y \quad (*)$.

$$\begin{aligned} f(x) &= f(x) - 2^{-n} + 2^{-n} \\ &< y' + 2^{-n} && (y' \in F^{\mathcal{R}}(x,n)) \\ &< y && (\text{assumption } (*)) \end{aligned}$$

Proof of (3.2) is similar to (3.1). □

Remark 3.2.4. Let $F : \mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$, and $x \in \mathbb{R}$. Then

- If $x \notin \mathbf{dom}(f)$, regardless of the value of b, we have

$$(F(x) < b)^{\mathcal{R}} = (F(x) > b)^{\mathcal{R}} = \emptyset.$$

- Assuming $x \in \mathbf{dom}(f)$, we have
$$(F(x) < b)^{\mathcal{R}} = \emptyset \iff f(x) \geq b,$$
and respectively,
$$(F(x) > b)^{\mathcal{R}} = \emptyset \iff f(x) \leq b.$$

Remark 3.2.5. Let $F : \mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$ and $x \in \mathbb{R}$. Then, Lemma 3.2.1 is particularly useful for getting an upperbound for $f(x)$ using the statement

$$\mathit{upper} := \mathsf{choose}\ (r : \mathsf{real}) : F(x) < r$$

and respectively, getting a lowerbound for $f(x)$ using the statement

$$\mathit{lower} := \mathsf{choose}\ (r : \mathsf{real}) : F(x) > r.$$

Definition 3.2.6. Let $F, G : \mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be **WhileCC**-procedures and $x_1, x_2 \in \mathbb{R}$. We define the **WhileCC**-procedure $F(-) < G(-)$ as a shorthand for

```
$(F(-) < G(-)) \equiv$ proc
                          in $x_1$ : real $x_2$ : real
                          aux $c$ : nat
                       begin
                          $c :=$ choose $(k : \mathsf{nat}) : F(x_1, k) + 2^{-k} < G(x_2, k) - 2^{-k}$
                          /* If terminates it means $f(x_1) < g(x_2)$ */
                          return $tt$
                       end
```

The semantics of the procedure above is

$$\begin{aligned}(F(x_1) < G(x_2))^{\mathcal{R}} = \{\mathsf{tt} \mid \exists k \in \mathbb{N}\ \exists y_1 \in F^{\mathcal{R}}(x_1, k)\ \exists y_2 \in F^{\mathcal{R}}(x_2, k) \\ y_1 + 2^{-k} < y_2 - 2^{-k}\}.\end{aligned}$$

Note that instead of $(F(-) < G(-))^{\mathcal{R}}(x_1, x_2)$ (resp. $(F(-) > G(-))^{\mathcal{R}}(x_1, x_2)$) we write $(F(x_1) < G(x_2))^{\mathcal{R}}$ (resp. $(F(x_1) > G(x_2))^{\mathcal{R}}$).

Lemma 3.2.7. Let $F, G : \mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be **WhileCC**-procedures approximating the functions $f, g : \mathbb{R} \to \mathbb{R}$ respectively, and $x_1, x_2 \in \mathbb{R}$. Then we have:

$$\begin{aligned} f(x_1) < g(x_2) \quad &\iff \exists n \in \mathbb{N} \quad \exists y_1 \in F^{\mathcal{R}}(x_1, n)\ \exists y_2 \in G^{\mathcal{R}}(x_2, n) \\ &\quad\ y_1 + 2^{-n} < y_2 - 2^{-n} \\ &\iff (F(x_1) < G(x_2))^{\mathcal{R}} = \{\mathsf{tt}\} \end{aligned}$$

Proof. The proof is similar to Lemma 3.2.3.

(⇒) If $f(x_1) < g(x_2)$ then $g(x_2) - f(x_1) > 0$, and there is $n \in \mathbb{N}$ such that

$$2^{-n} < (g(x_2) - f(x_1))/2.$$

Then, for any $y_1 \in F^{\mathcal{R}}(x_1, n)$ and any $y_2 \in G^{\mathcal{R}}(x_2, n)$ we have $f(x_1) - 2^{-n} < y_1 < f(x_1) + 2^{-n}$ and hence:

$$\begin{aligned} y_1 + 2^{-n} &< f(x_1) + 2^{-n+1} \\ &< f(x_1) + 2((g(x_2) - f(x_1))/2) \\ &= f(x_1) + g(x_2) - f(x_1) \\ &= g(x_2) \end{aligned}$$

($\Leftarrow$) We assume that $\exists n\ \exists y_1 \in F^{\mathcal{R}}(x_1, n)\ \exists y_2 \in G^{\mathcal{R}}(x_2, n)$ such that $y_1 + 2^{-n} < g(x_2) - 2^{-n}$ (*). Then:

$$\begin{aligned} f(x_1) &= f(x_1) - 2^{-n} + 2^{-n} \\ &< y_1 + 2^{-n} && (y_1 \in F^{\mathcal{R}}(x_1, n)) \\ &< g(x_2) - 2^{-n} && (\text{assumption } (*)) \\ &< g(x_2) \end{aligned}$$

$\square$

Remark 3.2.8. Let F, G : real $\times$ nat $\to$ real be **WhileCC**-procedures approximating the functions $f, g : \mathbb{R} \to \mathbb{R}$ respectively, and $x_1, x_2 \in \mathbb{R}$. Then

- If $x_1 \notin \mathbf{dom}(f)$ or $x_2 \notin \mathbf{dom}(g)$, regardless of the value of y, we have
$$(F(x_1) < G(x_2))^{\mathcal{R}} = (F(x_1) > G(x_2))^{\mathcal{R}} = \emptyset.$$
- Assuming $x_1 \in \mathbf{dom}(f)$ and $x_2 \in \mathbf{dom}(g)$, we have
$$(F(x_1) < G(x_2))^{\mathcal{R}} = \emptyset \iff f(x_1) \geq g(x_2),$$
and respectively,
$$(F(x_1) > G(x_2))^{\mathcal{R}} = \emptyset \iff f(x_1) \leq g(x_2).$$

3.3 Basic Functions

Theorem 3.3.1. Let c be an α-computable real. Then the constant function $f(x) = c$ is **WhileCC**-approximable.

Proof. Immediately follows from Lemma 2.4.8 and Equivalence Lemma 2.6.4. $\square$

Theorem 3.3.2. The identity function $id(x) = x$ is **WhileCC**-approximable.

Proof. Immediately follows from Lemma 2.4.9 and Equivalence Lemma 2.6.4. $\square$

3.4 Addition

In this section, we construct a **WhileCC**-procedure that approximates $(f+g)(x) = f(x) + g(x)$ using **WhileCC**-procedures that approximate the functions f and g.

Definition 3.4.1. Let F, G : real $\times$ nat $\to$ real be **WhileCC**-procedures approximating the functions $f, g : \mathbb{R} \to \mathbb{R}$ respectively. Then the **WhileCC**-procedure

```
(F + G) ≡ proc
             in x : real c : nat
             out res : real
          begin
             res := F(x, c + 1) + G(x, c + 1)
          end
```

of type real $\times$ nat $\to$ real has the semantics

$$(F+G)^{\mathcal{R}}(x, n) = \{y_1 + y_2 \mid y_1 \in F^{\mathcal{R}}(x, n+1) \wedge y_2 \in G^{\mathcal{R}}(x, n+1)\}.$$

Note that since we want the overall error of our approximation to be less than 2^{-c}, the idea is to define the procedure $(F+G)$ to approximate each of $f(x)$ and $g(x)$ with at most half the error bound i.e. $2^{-(c+1)}$, so that when adding the two values, the overall error would be smaller that 2^{-c}. We prove that the **WhileCC**-procedure $(F+G)$ approximates the function $(f+g)(x) = f(x) + g(x)$.

Lemma 3.4.2. Let F, G : $\mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be **WhileCC**-procedures approximating the functions $f, g : \mathbb{R} \to \mathbb{R}$ respectively, and $x \in \mathbf{dom}(f+g)$. Then we have $(F+G)^{\mathcal{R}}(x,n) \neq \emptyset$ for any $n \in \mathbb{N}$.

Proof. Let us assume $x \in \mathbf{dom}(f+g) = \mathbf{dom}(f) \cap \mathbf{dom}(g)$. Since F (resp. G) is the **WhileCC**-procedure approximating f (resp. g), There are $y_1 \in F^{\mathcal{R}}(x, n+1) \neq \emptyset$ and respectively $y_2 \in G^{\mathcal{R}}(x, n+1) \neq \emptyset$. This, by the definition of $(F+G)^{\mathcal{R}}$ means that $y_1 + y_2 \in (F+G)^{\mathcal{R}}(x,n)$, and hence $(F+G)^{\mathcal{R}}(x,n) \neq \emptyset$. □

Lemma 3.4.3. Let F, G : $\mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be **WhileCC**-procedures approximating the functions $f, g : \mathbb{R} \to \mathbb{R}$ respectively and $x \notin \mathbf{dom}(f+g)$. Then we have $\forall n \in \mathbb{N} \quad (F+G)^{\mathcal{R}}(x,n) = \emptyset$.

Proof. Assume $x \notin \mathbf{dom}(f+g)$. This means that at least $x \notin \mathbf{dom}(f)$ or $x \notin \mathbf{dom}(g)$. Hence at least one of the sets $F^{\mathcal{R}}(x, n+1)$ or $G^{\mathcal{R}}(x, n+1)$ is empty which means that $(F+G)^{\mathcal{R}}(x,n) = \emptyset$. □

Lemma 3.4.4. Let F, G : $\mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be **WhileCC**-procedures approximating the functions $f, g : \mathbb{R} \to \mathbb{R}$ respectively, and $n \in \mathbb{N}$. Then we have

$$(F+G)^{\mathcal{R}}(x,n) \subseteq \mathbf{Nbd}((f+g)(x), 2^{-n})$$

for all $x \in \mathbf{dom}(f+g)$.

Proof. By definition of $(F+G)^{\mathcal{R}}(x,n)$, for any $z \in (F+G)^{\mathcal{R}}(x,n)$ there are $y_1 \in F^{\mathcal{R}}(x, n+1)$ and $y_2 \in G^{\mathcal{R}}(x, n+1)$ such that $y_1 + y_2 = z$. Hence:

$$\begin{aligned}
&y_1 + y_2 = z \in (F+G)^{\mathcal{R}}(x,n) \\
\Longrightarrow\ &|f(x) - y_1| < 2^{-n-1} \ \wedge\ |g(x) - y_2| < 2^{-n-1} \\
\Longrightarrow\ &|f(x) + g(x) - y_1 - y_2| < 2^{-n} \\
\Longrightarrow\ &|(f+g)(x) - (y_1 + y_2)| < 2^{-n} \\
\Longrightarrow\ &y_1 + y_2 \in \mathbf{Nbd}((f+g)(x), 2^{-n}) \\
\Longrightarrow\ &z \in \mathbf{Nbd}((f+g)(x), 2^{-n})
\end{aligned}$$

□

Theorem 3.4.5. Let F, G : $\mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be **WhileCC**-procedures approximating the functions $f, g : \mathbb{R} \to \mathbb{R}$ respectively. Then the **WhileCC**-procedure $(F+G) : \mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ **WhileCC**-approximates $(f+g)(x) = f(x) + g(x)$.

Proof. Follows directly from Lemmas 3.4.2, 3.4.3, 3.4.4. □

3.5 Multiplication

In this section, we construct a **WhileCC**-procedure that approximates $(f \cdot g)(x) = f(x) \cdot g(x)$ using **WhileCC**-procedures that approximate the functions f and g.

Definition 3.5.1. Let F, G : real $\times$ nat $\to$ real be **WhileCC**-procedures approximating the functions $f, g : \mathbb{R} \to \mathbb{R}$ respectively. Then the **WhileCC**-procedure

```
(F · G) ≡ proc
              in $x$ : real $n$ : nat
              aux $m_1$ : nat $m_2$ : nat $y_1$ : real $y_2$ : real
          begin
              $m_1$ := choose ($k$ : nat) : $F(x) < 2^{-n-1+k}$ and $F(x) > -2^{-n-1+k}$
              $y_1 := G(x, m_1)$
              $m_2$ := choose ($k$ : nat) : ${y_1}^2 < 2^{-2n-2+k}$
              $y_2 := F(x, m_2)$
              return $y_1 \times y_2$
          end
```

of type real $\times$ nat $\to$ real has the semantics

$$(F \cdot G)^{\mathcal{R}}(x, n) = \{y_1 \cdot y_2 \mid \exists m_1, m_2 \in \mathbb{N} \quad y_1 \in G^{\mathcal{R}}(x, m_1) \wedge y_2 \in F^{\mathcal{R}}(x, m_2) \wedge |f(x)| < 2^{-n-1+m_1} \wedge y_1^2 < 2^{-2n-2+m_2}\}.$$

Note that the idea here is to compute "how precise" the approximations of $f(x)$ and $g(x)$ need to be, so that the product of the outputs of those approximations (y_1 and y_2) gives us the requested precision (2^{-n}):

Lemma 3.5.2. Let F, G : real $\times$ nat $\to$ real be **WhileCC**-procedures approximating the functions $f, g : \mathbb{R} \to \mathbb{R}$ respectively. Then for any $y_1 \in G^{\mathcal{R}}(x, m_1)$ and $y_2 \in F^{\mathcal{R}}(x, m_2)$ with

(i) $|f(x)| < 2^{-n-1+m_1}$, and

(ii) $y_1^2 < 2^{-2n-2+m_2}$

we have:

$$|y_1 y_2 - f(x)g(x)| < 2^{-n}.$$

Proof. To prove $|y_1 y_2 - f(x)g(x)| < 2^{-n}$, first we prove that

$$|y_1 y_2 - f(x)g(x)| < 2^{-m_2}|y_1| + 2^{-m_1}|f(x)|.$$

Then we need to show that each of the terms $2^{-m_2}|y_1|$ and $2^{-m_1}|f(x)|$ are less than 2^{-n-1}.

To show that $|y_1 y_2 - f(x)g(x)| < 2^{-m_2}|y_1| + 2^{-m_1}|f(x)|$:

$$\begin{aligned} |y_1 y_2 - f(x)g(x)| &= |y_1 y_2 - f(x)y_1 + f(x)y_1 - f(x)g(x)| \\ &\le |y_1 y_2 - f(x)y_1| + |f(x)y_1 - f(x)g(x)| \\ &= |y_1(y_2 - f(x))| + |f(x)(y_1 - g(x))| \\ &= |y_1||y_2 - f(x)| + |f(x)||y_1 - g(x)| \\ &< 2^{-m_2}|y_1| + 2^{-m_1}|f(x)|. \quad \text{(since } y_1 \in G^{\mathcal{R}}(x, m_1),\ y_2 \in F^{\mathcal{R}}(x, m_2)\text{)} \end{aligned}$$

Now to show that $2^{-m_2}|y_1| < 2^{-n-1}$, using (ii), we have:

$$\begin{aligned} y_1^2 < 2^{-2n-2+m_2} &\implies y_1^2 < 2^{-2n-2+2m_2} \\ &\implies |y_1| < 2^{-n-1+m_2} \\ &\implies 2^{-m_2}|y_1| < 2^{-n-1}. \end{aligned}$$

Now using (i), we have $|f(x)| < 2^{-n-1+m_1}$ and hence $2^{-m_1}|f(x)| < 2^{-n-1}$. This proves that $2^{-m_2}|y_1| + 2^{-m_1}|f(x)| < 2^{-n}$. □

Now, we go on to prove that the **WhileCC**-procedure $(F \cdot G)$ approximates the function $(f \cdot g)(x) = f(x) \cdot g(x)$.

Lemma 3.5.3. Let F, G : $\mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be **WhileCC**-procedures approximating the functions $f, g : \mathbb{R} \to \mathbb{R}$ respectively, and $x \in \mathbf{dom}(f \cdot g)$. Then we have $(F \cdot G)^{\mathcal{R}}(x, n) \neq \emptyset$ for any $n \in \mathbb{N}$.

Proof. Since $x \in \mathbf{dom}(f \cdot g) = \mathbf{dom}(f) \cap \mathbf{dom}(g)$ and $F^{\mathcal{R}}, G^{\mathcal{R}}$ approximate f, g respectively, for any $n \in \mathbb{N}$ we have $F^{\mathcal{R}}(x, n) \neq \emptyset$ and $G^{\mathcal{R}}(x, n) \neq \emptyset$. This means that since $x \in \mathbf{dom}(f)$, we can choose m_1 sufficiently large so that 2^{-n-1+m_1} is larger than $|f(x)|$. The trick here is to break the comparison to $|f(x)|$ into two comparisons: $f(x) < 2^{-n-1+m_1}$ and $f(x) > -2^{-n-1+m_1}$.
Now with fixed m_1 we can choose $y_1 \in G^{\mathcal{R}}(x, m_1)$ and we can find m_2 sufficiently large that $y_1^2 < 2^{-2n-2+m_2}$ and a $y_2 \in F^{\mathcal{R}}(x, m_2)$. Now, since multiplication is total, $y_1 y_2$ can be computed and hence $(F \cdot G)^{\mathcal{R}}(x, n) \neq \emptyset$.

□

Lemma 3.5.4. For any real $x \notin \mathbf{dom}(f \cdot g)$ and for any $n \in \mathbb{N}$, we have $(F \cdot G)^{\mathcal{R}}(x, n) = \emptyset$.

Proof. Assume $x \notin \mathbf{dom}(f \cdot g)$. This means that at least $x \notin \mathbf{dom}(f)$ or $x \notin \mathbf{dom}(g)$.

- If $x \notin \mathbf{dom}(f)$ then the comparison procedure $(F(-) < -)$ will not terminate and hence $(f \cdot g)(x, n) = \emptyset$.
- If $x \in \mathbf{dom}(f)$ but $x \notin \mathbf{dom}(g)$, then no $y_1 \in G^{\mathcal{R}}(x, m_1)$ can be found since $G^{\mathcal{R}}(x, m_1) = \emptyset$.

and hence $(F \cdot G)^{\mathcal{R}}(x, n) = \emptyset$ in either of the cases. □

Lemma 3.5.5. For any real $x \in \mathbf{dom}(f \cdot g)$ and for any $n \in \mathbb{N}$, we have $(F \cdot G)^{\mathcal{R}}(x, n) \subseteq \mathbf{Nbd}(f(x) \cdot g(x), 2^{-n})$.

Proof. Using the definition of the procedure $(F \cdot G)$, Lemma 3.5.2 immediately implies that $(F \cdot G)^{\mathcal{R}}(x, n) \subseteq \mathbf{Nbd}(f(x) \cdot g(x), 2^{-n})$. □

Theorem 3.5.6. Let F, G : $\mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be **WhileCC**-procedures approximating the functions $f, g : \mathbb{R} \to \mathbb{R}$ respectively. Then the **WhileCC**-procedure $(F \cdot G)$: $\mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ **WhileCC**-approximabtes $(f \cdot g)(x) = f(x) \cdot g(x)$.

Proof. Follows directly from Lemmas 3.5.3, 3.5.4, 3.5.5. □

3.6 Division

In this section, we construct a **WhileCC**-procedure that approximates $\mathrm{div}_f(x) = 1/f(x)$ using a **WhileCC**-procedure that approximates the functions f.

Definition 3.6.1. Let $F : \mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$, and and $x \in \mathbb{R}$. Then the **WhileCC**-procedure

```
$(1/F(-,-)) \equiv$ proc
                      in $x$ : real $n$ : nat
                      aux $res$ : real $chosenVal$ : nat $m$ : real
                    begin
                      $chosenVal$ := choose ($k$ : nat) : proc
                                                            in $k$ : nat $x$ : real
                                                            aux $res$ : bool
                                                          begin
                                                            if $k =_{\mathsf{nat}} 1$ then
                                                              $res := 0 < F(x)$
                                                            else
                                                              $res := F(x) < 0$
                                                            fi
                                                            return $res$
                                                          end
                      if $chosenVal =_{\mathsf{nat}} 1$ then
                        $res, m$ := choose ($q$ : real, $mVal$ : real) :
                                      abs$(q - \frac{1}{mVal}) < 2^{-n}$
                                        and
                                      $(0 < mVal < F(x))$
                      else
                        $res, m$ := choose ($q$ : real, $mVal$ : real) :
                                      abs$(q - \frac{1}{mVal}) < 2^{-n}$
                                        and
                                      $(F(x) < mVal < 0)$
                      fi
                      return $res$
                    end
```

of type $\mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ has the semantics

$$\begin{aligned}(1/F(x,n))^{\mathcal{R}} = \{q \in \mathbb{Q} \mid \exists m \in \mathbb{Q} \quad & 0 < mid < f(x) \quad \wedge 0 < |q - (1/m)| < 2^{-n}\} \\ \cup \quad & \\ \{q \in \mathbb{Q} \mid \exists m \in \mathbb{Q} \quad & f(x) < m < 0 \qquad \wedge 0 < |q - (1/m)| < 2^{-n}\}.\end{aligned}$$

Note that instead of $(1/F(-,-))^{\mathcal{R}}(x,n)$ we write $(1/F(x,n))^{\mathcal{R}}$.

We handle the case of $0 < f(x)$ and $f(x) < 0$ separately. In each case we choose a rational value q that is sufficiently close to $1/f(x)$. Since we cannot calculate the exact value of $f(x)$, we choose an approximation of it called m here and choose q to be sufficiently close to m.

Lemma 3.6.2. Let $f : \mathbb{R} \to \mathbb{R}$. Then for any $q, m \in \mathbb{Q}$ we have

(i) $0 < m < f(x) \wedge 0 < |q - (1/m)| < 2^{-n} \implies |q - (1/f(x))| < 2^{-n}$, and

(ii) $f(x) < m < 0 \wedge 0 < |q - (1/m)| < 2^{-n} \implies |q - (1/f(x))| < 2^{-n}$.

Proof. First let's assume $f(x) > 0$, then

$$\begin{aligned} & 0 < |q - 1/m| < 2^{-n} \\ \implies & |q - 1/f(x)| < 2^{-n} & \text{(since } 0 < mid < f(x)) \end{aligned}$$

Now the case where $f(x) < 0$ is similar: we have

$$\begin{aligned} & 0 < |q - 1/m| < 2^{-n} \\ \implies & |q - 1/f(x)| < 2^{-n} & \text{(since } f(x) < mid < 0) \end{aligned}$$

□

Now, we go on to prove that the **WhileCC**-procedure $(1/F(-,-))$ approximates the function $\text{div}_f(x) = 1/f(x)$.

Lemma 3.6.3. Let F : $\mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$. Then for any real $x \in \mathbf{dom}(\text{div}_f)$ and for any $n \in \mathbb{N}$, we have $(1/F(x,n))^{\mathcal{R}} \neq \emptyset$.

Proof. Since $x \in \mathbf{dom}(\text{div}_f)$. This means that $x \in \mathbf{dom}(f)$ and $f(x) \neq 0$. This means there is always a rational number m between 0 and $f(x)$. Now no matter what value the chosen non-zero mid has, we can choose a rational number q sufficiently close to $1/m$ satisfying $|q - 1/mid| < 2^{-n}$. Hence $(1/F(x,n))^{\mathcal{R}} \neq \emptyset$. □

Lemma 3.6.4. Let F : $\mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$. Then for any real $x \notin \mathbf{dom}(\text{div}_f)$ and for any $n \in \mathbb{N}$, we have $(1/F(x,n))^{\mathcal{R}} = \emptyset$.

Proof. Assume $x \notin \mathbf{dom}(\text{div}_f)$. This means that at least $x \notin \mathbf{dom}(f)$ or $f(x) = 0$. If $x \notin \mathbf{dom}(f)$, then $(F(x) > 0)^{\mathcal{R}} = (F(x) < q)^{\mathcal{R}} = \emptyset$, and hence $(1/F(x,n))^{\mathcal{R}} = \emptyset$. Now let's assume $x \in \mathbf{dom}(f)$ and $f(x) = 0$. This means no $0 < mid < 0$ can be found and hence $(1/F(x,n))^{\mathcal{R}} = \emptyset$. □

Lemma 3.6.5. Let F : $\mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$. Then $\forall x \in \mathbf{dom}(f)$ with $f(x) \neq 0$, $\forall n \in \mathbb{N}$, $\forall q \in (1/F(x,n))^{\mathcal{R}}$:

$$|1/f(x) - q| < 2^{-n}.$$

Proof. Using the definition of the procedure $(1/F(-,-))$, Lemma 3.6.2 immediately implies that $(1/F(x,n))^{\mathcal{R}} \subseteq \mathbf{Nbd}(1/f(x), 2^{-n})$. □

Theorem 3.6.6. Let F : $\mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$. Then the **WhileCC**-procedure $(1/F(-,-))$ approximates div_f.

Proof. Follows directly from Lemmas 3.6.3, 3.6.4, 3.6.5. □

3.7 Exponential

In this section, we discuss **WhileCC**-approximability of the function $\exp_f(x)$ for a **WhileCC**-approximable function f. First, we define a **WhileCC**-program that approximates $\exp(x)$ and then based on that, assuming we have a **WhileCC**-procedure for approximating the function f, we build another **WhileCC**-program that approximates $\exp_f(x) = \exp(f(x))$.

3.7.1 WhileCC-Approximability of exp(x)

Since the function e^x is analytic on $\mathbb{R}$, we can compute sufficiently many terms of its Taylor series to get a 'sufficiently close' approximation. The Taylor expansion of e^x is as follows:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} \tag{3.3}$$

To approximate e^x, we need to figure out how many iterations of the summation above need to be calculated to achieve the desired precision. Let us define the **WhileCC**-procedure isProperIndex : nat $\times$ real $\times$ nat $\to$ real below:

```
isProperIndex ≡ proc
    in $N$ : nat $x$ : real $c$ : nat
    aux $res$ : bool
begin
    $res := x^N/\mathsf{Factorial}(N) < (2^{-c})$
          and $(2 \times x) < N$
          and $(2 \times -x) < N$
    return $res$
end
```

The semantics of the procedure isProperIndex is as below:

$$\mathsf{isProperIndex}^{\mathcal{R}}(N, x, c) = \{\mathsf{tt}\} \iff x^n/N! < 2^{-c} \ \land \ 2|x| < N.$$

The procedure isProperIndex returns true only if calculating the first n terms of the series will produce a sufficiently precise approximation.

Lemma 3.7.1. For all $N \in \mathbb{N}$, $x \in \mathbb{R}$, and $c \in \mathbb{N}$, we have

$$\mathsf{isProperIndex}^{\mathcal{R}}(N, x, c) = \{\mathsf{tt}\} \implies \left| e^x - \sum_{n=0}^{N} \frac{x^n}{n!} \right| < 2^{-c}$$

Proof. Let us assume isProperIndex$^{\mathcal{R}}(N, x, c) = \{\mathsf{tt}\}$. By definition, this means that

(i) $x^n/N! < 2^{-c}$, and

(ii) $2|x| < N$

We compute an upper bound for the error of the Taylor series approximation up to the

Nth term:

$$\begin{aligned}
\left| e^x - \sum_{n=0}^{N} \frac{x^n}{n!} \right| &= \left| \sum_{n=N+1}^{\infty} \frac{x^n}{n!} \right| \\
&= \left| \sum_{k=1}^{\infty} \frac{x^{N+k}}{(N+k)!} \right| \\
&\leq \sum_{k=1}^{\infty} \frac{|x|^{N+k}}{(N+k)!} && \text{(by triangle inequality)} \\
&< \sum_{k=1}^{\infty} \frac{1}{2^k} \cdot \frac{|x|^N}{N!} && \text{(by assumption (ii) and Lemma A.0.4)} \\
&= 1 \cdot \frac{|x|^N}{N!} \\
&< 2^{-c} && \text{(by assumption (i))}
\end{aligned}$$

□

Now we need to make sure that the premise in Lemma 3.7.1 can actually be satisfied.

Lemma 3.7.2. For any $c \in \mathbb{N}$ and arbitrary $x \in \mathbb{R}$, there is some $N \in \mathbb{N}$ for which

$$\mathsf{isProperIndex}^{\mathcal{R}}(N, x, c) = \{\mathsf{tt}\}.$$

Proof. By Theorem A.0.1, choosing $\epsilon = 2^{-c}$, there exists an index $N > 2|x|$ such that $\frac{x^N}{N!} < 2^{-c}$. □

Now we know that for arbitrary inputs x, we can find the necessary number of terms we must calculate to get the desired precision.

Definition 3.7.3. The **WhileCC**-procedure

```
Exp ≡ proc
        in $x$ : real $c$ : nat
        aux $counter$ : nat $N$ : nat $sum$ : real
      begin
        $counter := 0_{\mathsf{nat}}$
        $sum := 0_{\mathsf{real}}$
        $N :=$ choose ($k$ : nat) : isProperIndex($k, x, c$)
        while $counter <_{\mathsf{nat}} N$ do
          $sum := sum + x^{counter}/\mathsf{Factorial}(counter)$
          $counter := counter + 1$
        od
        return $sum$
      end
```

of type real × nat → real has the semantics

$$\mathsf{Exp}^{\mathcal{R}}(x, c) = \left\{ \sum_{n=0}^{N} \frac{x^n}{n!} \mid \mathsf{isProperIndex}(N, x, c) \right\}$$

The procedure Exp approximates e^x for $x \in \mathbb{R}$ by calculating the first N terms of the series (3.3) where N is chosen to satisfy isProperIndex.

Lemma 3.7.4. For any real $x \in \mathbb{R}$ and for any $n \in \mathbb{N}$, we have $\mathsf{Exp}^{\mathcal{R}}(x, n) \neq \emptyset$.

Proof. Follows directly from the definition of $\mathsf{Exp}^{\mathcal{R}}$ and Lemma 3.7.2. □

Lemma 3.7.5. For any $x \in \mathbf{dom}(\exp) = \mathbb{R}$, $n \in \mathbb{N}$, and $y \in \mathsf{Exp}^{\mathcal{R}}(x, n)$ we have

$$|e^x - y| < 2^{-n}.$$

Proof. Follows directly from the definition of $\mathsf{Exp}^{\mathcal{R}}$ and Lemma 3.7.1. □

Theorem 3.7.6. The **WhileCC**-procedure Exp given in Definition 3.7.3 approximates exp.

Proof. Follows directly from Lemmas 3.7.4, 3.7.5 and the fact that $\mathbf{dom}(\exp) = \mathbb{R}$. □

3.7.2 WhileCC-Approximability of exp(f(x))

Definition 3.7.7. Let F : real × nat → real be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$, and $x \in \mathbb{R}$. Then the **WhileCC**-procedure

```
$\mathsf{Exp}_F \equiv$ proc
    in $n$ : nat $x$ : real
    aux $res$ : real $m$ : nat $z$ : nat
  begin
    $z :=$ choose $(k : nat) : F(x) < k$
    $m :=$ choose $(k : nat) : \mathsf{Exp}(2^{-k}) < 2^{-n-1-2z} + 1$
    $res := \mathsf{Exp}(F(x, m), n + 1)$
    return $res$
  end
```

of type real × nat → real has the semantics

$$\begin{aligned}\mathsf{Exp}_F^{\mathcal{R}}(x, n) = \{y \in \mathsf{Exp}^{\mathcal{R}}(r, n+1) \mid \\ &\exists z \in \mathbb{N} \quad f(x) < z \wedge \\ &\exists m \in \mathbb{N} \quad e^{(2^{-m})} < 2^{-n-1-2z} + 1 \wedge \\ &r \in F^{\mathcal{R}}(x, m)\}\end{aligned}$$

Intuitively, Exp_F approximates $\exp(f(x))$ by invoking $\mathsf{Exp}(F(x, m), n+1)$, where m is sufficiently large to ensure

$$\left|e^{f(x)} - y\right| < 2^{-n}$$

for any output y of $\mathsf{Exp}(F(x, m), n + 1)$.

Lemma 3.7.8. Let the **WhileCC**-procedure F : real × nat → real approximate $f : \mathbb{R} \to \mathbb{R}$ and $x \in \mathbf{dom}(f)$. Now let $m, z \in \mathbb{N}$, $r \in F^{\mathcal{R}}(x, m)$, and $y \in \mathsf{Exp}^{\mathcal{R}}(r, n + 1)$ satisfy the two conditions

(i) $e^{(2^{-m})} < 2^{-n-1-2z} + 1$, and

(ii) $f(x) < z$.

Then

$$\left|e^{f(x)} - y\right| < 2^{-n}.$$

Proof. First, we can see that

$$\begin{aligned}\left|e^{f(x)} - y\right| &= \left|e^{f(x)} - e^r + e^r - y\right| \\ &\leq \left|e^{f(x)} - e^r\right| + \left|e^r - y\right|.\end{aligned}$$

Now, to prove $\left|e^{f(x)} - y\right| < 2^{-n}$ it suffices to prove $|e^r - y| < 2^{-n-1}$ and $\left|e^{f(x)} - e^r\right| < 2^{-n-1}$.

(i) $|e^r - y| < 2^{-n-1}$: This is easily seen by assumption $y \in \mathsf{Exp}^{\mathcal{R}}(r, n+1)$.

(ii) $\left|e^{f(x)} - e^r\right| < 2^{-n-1}$: First observe the following:

$$\begin{aligned}\left|e^{f(x)} - e^r\right| &\leq \left|e^{f(x)} - e^{f(x)+2^{-m}}\right| && \text{(by Lemma A.0.3)} \\ &= \left|e^{f(x)} - e^{f(x)}e^{2^{-m}}\right| \\ &= \left|e^{f(x)}(1 - e^{2^{-m}})\right| \\ &= \left|e^{f(x)}\right|\left|(1 - e^{2^{-m}})\right| \\ &= e^{f(x)}(e^{2^{-m}} - 1) && \text{(since } e^{2^{-m}} \geq 1) \\ &< e^z(e^{2^{-m}} - 1) && \text{(by (ii))} \\ &< 4^z(e^{2^{-m}} - 1) \\ &= 2^{2z}(e^{2^{-m}} - 1)\end{aligned}$$

Now using (i), we have

$$\begin{aligned}e^{2^{-m}} < 2^{-n-1-2z} + 1 &\implies e^{2^{-m}} - 1 < 2^{-n-1-2z} \\ &\implies e^{2^{-m}} - 1 < 2^{-n-1} \cdot 2^{-2z} \\ &\implies 2^{2z}(e^{2^{-m}} - 1) < 2^{-n-1}\end{aligned}$$

which implies $\left|e^{f(x)} - e^r\right| < 2^{-n-1}$.

□

Now we prove that the **WhileCC**-procedure $\mathsf{Exp}_F^{\mathcal{R}}$ approximates $\exp_f(x)$.

Lemma 3.7.9. For any real $x \in \mathbf{dom}(e^f)$ and for any $n \in \mathbb{N}$, we have $\mathsf{Exp}_F^{\mathcal{R}}(x, n) \neq \emptyset$.

Proof. Assuming $x \in \mathbf{dom}(\exp_f)$, this means $x \in \mathbf{dom}(f)$. Since F approximates $f(x)$, we can find a $z > f(x)$. Now looking at the constraint $e^{(2^{-m})} < 2^{-n-1-2z} + 1$, we know that $2^{-n-1-2z} > 0$ and hence $2^{-n-1-2z} + 1 > 1$. This means we can find an index m sufficiently large so that $e^{(2^{-m})} < 2^{-n-1-2z} + 1$ since $\lim_{m\to\infty} e^{(2^{-m})} = 1$. Now since $x \in \mathbf{dom}(f)$, we know that $F^{\mathcal{R}}(x, m) \neq \emptyset$ so we can choose an $r \in F^{\mathcal{R}}(x, m)$. We also know that $\mathbf{dom}(\exp) = \mathbb{R}$, so $\mathsf{Exp}_F^{\mathcal{R}} \neq \emptyset$ and hence $\mathsf{Exp}_F^{\mathcal{R}}(x, n) \neq \emptyset$. □

Lemma 3.7.10. For any real $x \notin \mathbf{dom}(e^{f(x)})$ and $n \in \mathbb{N}$, we have $\mathsf{Exp}_F^{\mathcal{R}}(x, n) = \emptyset$.

Proof. Assuming $x \notin \mathbf{dom}(e^{f(x)})$, this means $x \notin \mathbf{dom}(f)$. So we have $F^{\mathcal{R}}(x, m) = \emptyset$. This shows that $F(x) < z$ does not terminate and hence $\mathsf{Exp}_F^{\mathcal{R}}(x, n) = \emptyset$ for any $n \in \mathbb{N}$. □

Lemma 3.7.11. Let F : real × nat → real be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$. Then $\forall x \in \mathbf{dom}(f)$, $\forall n \in \mathbb{N}$, $\forall y \in \mathsf{Exp}_F^{\mathcal{R}}(x, n)$:

$$\left| e^{f(x)} - y \right| < 2^{-n}.$$

Proof. Follows directly from Lemma 3.7.8. □

Theorem 3.7.12. Let F : real × nat → real be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$. Then the **WhileCC**-procedure $\mathsf{Exp}_F^{\mathcal{R}}$ approximates $\exp_f(x)$.

Proof. Follows directly from Lemmas 3.7.9, 3.7.10, 3.7.11. □

3.8 Logarithm

In this section, we discuss **WhileCC**-approximability of the function $\ln(f(x))$ for any **WhileCC**-approximable function f.

Since we already have a way of approximating the inverse of $\ln(x)$, which is $\exp(x)$, we can use it to construct a **WhileCC**-procedure that approximates $\ln(f(x))$ for a **WhileCC**-approximable function f.

Definition 3.8.1. Let F : real × nat → real be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$, and and $x \in \mathbb{R}$. Then the **WhileCC**-procedure

```
Ln_F ≡ proc
    in $x$ : real $n$ : nat
    aux $res$ : real
begin
    $res$ := choose ($y$ : real) : Exp($y$) < $F(x)$ and $F(x)$ < Exp($y + 2^{-n}$)
    return $res$
end
```

of type real × nat → real has the semantics

$$\mathsf{Ln}_F^{\mathcal{R}}(x, n) = \{y \in \mathbb{Q} \mid \ \exp(y) < f(x) < \exp(y + 2^{-n})\}$$

The **WhileCC**-procedure Ln_F approximates $\ln_f(x)$.

Lemma 3.8.2. Let F : real × nat → real be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$, and $x \in \mathbf{dom}(\ln_f)$. Then for any $n \in \mathbb{N}$, we have $\mathsf{Ln}_F^{\mathcal{R}}(x, n) \neq \emptyset$.

Proof. Since $x \in \mathbf{dom}(\ln_f)$, then $f(x) > 0$. Then there must be some $y \in \mathbb{Q}$ satisfying $\exp(y) < f(x)$, and we can choose y such that $\exp(y)$ is arbitrarily close to $f(x)$. Since exp is monotonically increasing, we can choose y to satisfy $f(x) < \exp(y+2^{-n})$. Therefore $\mathsf{Ln}_F^{\mathcal{R}}(x, n) \neq \emptyset$. □

Lemma 3.8.3. Let F : real × nat → real be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$. For any real $x \notin \mathbf{dom}(\ln_f)$ and $n \in \mathbb{N}$, we have $\mathsf{Ln}_F^{\mathcal{R}}(x, n) = \emptyset$.

Proof. Since $x \notin \mathbf{dom}(\ln_f)$, it means that at least $x \notin \mathbf{dom}(f)$, or $f(x) \leq 0$. In the case $x \notin \mathbf{dom}(f)$, the procedure $\mathsf{Exp}(y) < F(x)$ does not terminate and hence $\mathsf{Ln}_F^{\mathcal{R}}(x, n) = \emptyset$. In the case where $f(x)$ is defined but $f(x) \leq 0$, since $\exp(y)$ is positive, $\mathsf{Exp}(y) < F(x)$ cannot terminate and hence $\mathsf{Ln}_F^{\mathcal{R}}(x, n) = \emptyset$. □

Lemma 3.8.4. Let $F : \mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$. Then $\forall x \in \mathbf{dom}(f),\ \forall n \in \mathbb{N},\ \forall y \in \mathsf{Ln}_F^{\mathcal{R}}(x, n)$:

$$|\ln(f(x)) - y| < 2^{-n}.$$

Proof. Using the definition of Ln_F, we have

$$\begin{aligned} \exp(y) < f(x) < \exp(y + 2^{-n}) &\implies y < \ln(f(x)) < y + 2^{-n} \\ &\implies |\ln(f(x)) - y| < 2^{-n}. \end{aligned}$$

□

Theorem 3.8.5. Let $F : \mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$. Then $\mathsf{Ln}_F^{\mathcal{R}}$ **WhileCC**-approximates the function $\ln_f$.

Proof. Follows from Lemmas 3.8.2, 3.8.3, 3.8.4. □

3.9 Sine Function

In this section, we discuss **WhileCC**-approximability of the function $\sin_f(x)$ for a **WhileCC**-approximable function f. First, we define a **WhileCC**-program that approximates $\sin(x)$ and then based on that, assuming we have a **WhileCC**-procedure for approximating the function f, we build another **WhileCC**-program, that approximates $\sin(f(x))$.

3.9.1 WhileCC-Approximability of sin(x)

Since the function $\sin(x)$ is analytic on $\mathbb{R}$, we can compute sufficiently many terms of its Taylor series to get a "sufficiently close" approximation. Considering the Taylor expansion of $\sin(x)$ expanded around $x = 0$, we get:

$$\sin(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} \tag{3.4}$$

To approximate $\sin(x)$, we need to figure out how many iterations of the summation above need to be calculated to achieve the desired precision. Let us define the **WhileCC**-procedure $\mathsf{isProperIndex} : \mathsf{nat} \times \mathsf{real} \times \mathsf{nat} \to \mathsf{bool}$ below:

```
isProperIndex ≡ proc
                    in $N$ : nat $x$ : real $c$ : nat
                    out $r$ : bool
                begin
                    $r := x^N / \mathsf{Factorial}(N) <_{\mathsf{real}} 2^{-c}$
                          $\wedge\ 2 \times x < N$
                          $\wedge\ 2 \times -x < N$
                    return $r$
                end
```

The semantics of the procedure $\mathsf{isProperIndex}$ is as below:

$$\mathsf{isProperIndex}^{\mathcal{R}}(N, x, c) = \{\mathsf{tt}\} \iff x^N/N! < 2^{-c} \ \wedge \ 2|x| < N.$$

The procedure $\mathsf{isProperIndex}$ returns true only if calculating the first N terms of the series will produce a sufficiently precise approximation.

Lemma 3.9.1. Let $c \in \mathbb{N}$ and $x \in \mathbb{R}$. Then for any $N \in \mathbb{N}$

$$\mathsf{isProperIndex}^{\mathcal{R}}(N, x, c) = \{\mathsf{tt}\} \implies \left| \sin(x) - \sum_{n=0}^{N} (-1)^n \frac{x^{2n+1}}{(2n+1)!} \right| < 2^{-c}.$$

Proof. Let us assume $\mathsf{isProperIndex}^{\mathcal{R}}(N, x, c) = \{\mathsf{tt}\}$. By definition, this means that

(i) $x^N/N! < 2^{-c}$, and

(ii) $2|x| < N$.

Then,

$$\begin{aligned}
\left| \sin(x) - \sum_{n=0}^{N} (-1)^n \frac{x^{2n+1}}{(2n+1)!} \right| &= \left| \sum_{n=N+1}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} \right| \\
&= \left| \sum_{k=1}^{\infty} (-1)^{n+k} \frac{x^{2N+2k+1}}{(2N+2k+1)!} \right| \\
&\leq \sum_{k=1}^{\infty} \frac{|x|^{2N+2k+1}}{(2N+2k+1)!} && \text{(by triangle inequality)} \\
&< \sum_{k=1}^{\infty} \frac{1}{2^{N+2k+1}} \cdot \frac{|x|^N}{N!} \\
& && \text{(by assumption (ii) and Lemma A.0.4)} \\
&< \sum_{k=1}^{\infty} \frac{1}{2^k} \cdot \frac{|x|^N}{N!} \\
&= 1 \cdot \frac{|x|^N}{N!} \\
&< 2^{-c} && \text{(by assumption (i))}
\end{aligned}$$

□

Lemma 3.9.2. For any $c \in \mathbb{N}$ and arbitrary $x \in \mathbb{R}$, there is some $N \in \mathbb{N}$ for which

$$\mathsf{isProperIndex}^{\mathcal{R}}(N, x, c) = \{\mathsf{tt}\}.$$

Proof. By Theorem A.0.1, choosing $\epsilon = 2^{-c}$, there exists an index $N > 2|x|$ sufficiently large such that $x^N/N! < 2^{-c}$. □

Now we know that for arbitrary inputs $x \in \mathbb{R}$, we *can* find the necessary number of terms to calculate to get the desired precision.

Definition 3.9.3. The **WhileCC**-procedure

```
Sine ≡ proc
    in $x$ : real $c$ : nat
    aux $counter$ : nat $N$ : nat $sum$ : real
  begin
    $counter := 0_{\mathsf{nat}}$
    $sum := 0_{\mathsf{real}}$
    $N :=$ choose $(k : \mathsf{nat}) : \mathsf{isProperIndex}(k, x, c)$
    while $counter <_{\mathsf{nat}} N$ do
      $sum := sum +$
              $(-1^{counter} \times x^{2\times counter+1} / \mathsf{Factorial}(2 \times counter + 1))$
      $counter := counter + 1$
    od
    return $sum$
  end
```

of type real × nat → real has the semantics

$$\mathsf{Sine}^{\mathcal{R}}(x, c) = \left\{ \sum_{n=0}^{N} \frac{x^{2n+1}}{(2n+1)!} \mid \mathsf{isProperIndex}(N, x, c) = \{\mathsf{tt}\} \right\}.$$

The procedure Sine approximates $\sin(x)$ for any $x \in \mathbb{R}$ by calculating the first N terms of the series 3.4 where N is chosen to satisfy isProperIndex.

Corollary 3.9.4. For any $c \in \mathbb{N}$ and $x \in \mathbb{R}$, Using Lemma 3.9.2 and Definition 3.9.3, it immediately follows that

$$\mathsf{Sine}^{\mathcal{R}}(x, c) \neq \emptyset.$$

Lemma 3.9.5. For any $n \in \mathbb{N}$ and $x \in \mathbb{R}$ we have

$$\mathsf{Sine}^{\mathcal{R}}(x, n) \subseteq \mathbf{Nbd}(\sin(x), 2^{-n})$$

Proof. Follows immediately from Definition 3.9.3 and Lemma 3.9.1. □

3.9.2 WhileCC-Approximability of sin(f(x))

Here, we build a **WhileCC**-procedure that approximates $\sin(f(x))$ using the **WhileCC**-procedure Sine we constructed in the previous section.

Definition 3.9.6. Let F : real × nat → real be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$, and and $x \in \mathbb{R}$. Then the **WhileCC**-procedure

```
Sine_F ≡ proc
    in $x$ : real $n$ : nat
  begin
    return $\mathsf{Sine}(F(x, n+1), n+1)$
  end
```

of type real × nat → real has the semantics

$$\mathsf{Sine}_F^{\mathcal{R}}(x, n) \overset{def}{=} \{y \in \mathsf{Sine}^{\mathcal{R}}(r, n+1) \mid r \in F^{\mathcal{R}}(x, n+1)\}.$$

Let us prove that Sine_F approximates $\sin(f(x))$:

Lemma 3.9.7. For any real $x \in \mathbf{dom}(\sin(f))$ and for any $n \in \mathbb{N}$, we have $\mathsf{Sine}_F^{\mathcal{R}}(x, n) \neq \emptyset$.

Proof. Assuming $x \in \mathbf{dom}(\sin(f(x)))$, this means $x \in \mathbf{dom}(f)$. Since $x \in \mathbf{dom}(f)$, we know that $F^{\mathcal{R}}(x, n+1) \neq \emptyset$. So for any choice of $r \in F^{\mathcal{R}}(x, n+1)$, since we know that $\mathbf{dom}(\sin) = \mathbb{R}$, there exists a $y \in \mathsf{Sine}^{\mathcal{R}}(r, n+1)$. Hence, we have $\mathsf{Sine}_F^{\mathcal{R}}(x, n) \neq \emptyset$. □

Lemma 3.9.8. For any real $x \notin \mathbf{dom}(\sin(f))$ and $n \in \mathbb{N}$, we have $\mathsf{Sine}_F^{\mathcal{R}}(x, n) = \emptyset$.

Proof. Since F approximates f and $x \notin \mathbf{dom}(f)$, $F^{\mathcal{R}}(x, n+1)$ is empty and hence no $r \in F^{\mathcal{R}}(x, n+1)$ can be chosen which means $\mathsf{Sine}_F^{\mathcal{R}}(x, n) = \emptyset$. □

Lemma 3.9.9. Let $F : \mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$, then $\forall x \in \mathbf{dom}(f)$, $\forall n \in \mathbb{N}$, $\forall y \in \mathsf{Sine}_F^{\mathcal{R}}(x, n)$:

$$|\sin(f(x)) - y| < 2^{-n}.$$

Proof. For any **WhileCC**-prodecure approximating $f : \mathbb{R} \to \mathbb{R}$, arbitrary $x \in \mathbf{dom}(f)$ and $n \in \mathbb{N}$, and for any arbitrary $y \in \mathsf{Sine}_F^{\mathcal{R}}(x, n)$, we have $y \in \mathsf{Sine}^{\mathcal{R}}(r, n+1)$ where $r \in F^{\mathcal{R}}(x, n+1)$. Now, let us calculate the error of the approximator function below:

$$\begin{aligned}
& |\sin(f(x)) - y| \\
= {} & |\sin(f(x)) - \sin(r) + \sin(r) - y| \\
\leq {} & |\sin(f(x)) - \sin(r)| + |\sin(r) - y| \\
\leq {} & |f(x) - r| + |\sin(r) - y| && \text{(by the mean value theorem)} \\
< {} & 2^{-n-1} + |\sin(r) - y| && (\text{since } r \in F^{\mathcal{R}}(x, n+1)) \\
< {} & 2^{-n-1} + 2^{-n-1} \;=\; 2^{-n} && (\text{since } y \in \mathsf{Sine}_F^{\mathcal{R}}(r, n+1))
\end{aligned}$$

□

Theorem 3.9.10. Let $F : \mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$. Then the **WhileCC**-procedure $\mathsf{Sine}_F^{\mathcal{R}}$ approximates $\sin_f(x)$.

Proof. Follows from Lemmas 3.9.7, 3.9.8, 3.9.9. □

3.10 Arcsine

As mentioned in Definition 3.1.1, we would like to modify the common definition of $\arcsin(x)$ to a totalized version

$$\arcsin'(x) \overset{def}{=} \begin{cases} \frac{\pi}{2} & \text{if } x > 1 \\ \arcsin(x) & \text{if } -1 < x < 1 \\ -\frac{\pi}{2} & \text{if } x < -1 \end{cases}$$

to obtain a continuous function with an open domain. In this section, the ultimate goal is to prove **WhileCC**-approximability of the function $\arcsin'(f(x))$ for any **WhileCC**-approximable function f.

Starting with $\arcsin(x)$, in order to approximate $\arcsin'(x)$, we cannot simply compare the value of x against the endpoints and define constant functions outside the borders of the interval $(-1, 1)$. Intuitively, since our definition of comparison ($x < -1$ and $x > 1$)

is "partial" and undefined on $x = -1, 1$, we would have undefined holes in $x = -1, 1$.

The idea here would be to find overlapping intervals and approximate functions defined on those overlapping intervals using the choose operator.

In order to create "overlapping" intervals, we need to compute bounds such that approximating arcsine of any value closer than the bounds to the endpoints($x = -1, 1$) would be a "good-enough" approximation of arcsine of the endpoints $-1, 1$.

Let us define the **WhileCC**-procedure isCloseEnough of type real $\times$ nat $\to$ bool as

```
isCloseEnough ≡ proc
                  in $y$ : real $k$ : nat
                  aux $n$ : nat
                begin
                  $b := ((\mathsf{Sine} \cdot \mathsf{Sine})(2^{-k})) > 1 - y \times y$
                  return $b$
                end
```

with the semantics

$$\mathsf{isCloseEnough}^{\mathcal{R}}(y, k) = ((\mathsf{Sine} \cdot \mathsf{Sine})(2^{-k}) > 1 - y^2)^{\mathcal{R}}$$

of type real $\times$ nat $\to$ bool. This is the bound computed for y being a "good-enough" of approximation for the endpoints calculated below:

Lemma 3.10.1 (Bounds for arcsin). For any $k \in \mathbb{N}$ and any $y \in \mathbb{R}$:

1. $0 < y < 1 \ \wedge \ \mathsf{isCloseEnough}^{\mathcal{R}}(y, k) = \{\mathsf{tt}\} \implies \left|\frac{\pi}{2} - \arcsin(y)\right| < 2^{-k}$.
2. $-1 < y < 0 \ \wedge \ \mathsf{isCloseEnough}^{\mathcal{R}}(y, k) = \{\mathsf{tt}\} \implies \left|-\frac{\pi}{2} - \arcsin(y)\right| < 2^{-k}$.

Proof. 1. Let us assume $k \in \mathbb{N}$ and $0 < y < 1$ and hence $0 < \arcsin(y) < \frac{\pi}{2}$:

$$\begin{aligned}
& \mathsf{isCloseEnough}^{\mathcal{R}}(y, k) = \{\mathsf{tt}\} \\
\iff & 1 - y^2 < \sin^2(2^{-k}) \\
\implies & \sqrt{1 - y^2} < \sin(2^{-k}) && \text{(since } 0 < y < 1\text{)} \\
\iff & \cos(\arcsin(y)) < \sin(2^{-k}) \\
\iff & \sin(\frac{\pi}{2}) \cos(\arcsin(y)) < \sin(2^{-k}) \\
\iff & \sin(\frac{\pi}{2}) \cos(\arcsin(y)) - \cos(\frac{\pi}{2}) \sin(\arcsin(y)) < \sin(2^{-k}) \\
\iff & \sin(\frac{\pi}{2} - \arcsin(y)) < \sin(2^{-k}) \\
\iff & \frac{\pi}{2} - \arcsin(y) < 2^{-k} && \text{(sin is strictly increasing on } [0, \tfrac{\pi}{2}]\text{)} \\
\implies & \left|\frac{\pi}{2} - \arcsin(y)\right| < 2^{-k}
\end{aligned}$$

2. Similar to the first part, we assume $k \in \mathbb{N}$ and $-1 < y < 0$ and hence $-\frac{\pi}{2} <$

$\arcsin(y) < 0$:

$$\begin{aligned}
& \mathsf{isCloseEnough}^{\mathcal{R}}(y,k) = \{\mathsf{tt}\} \\
\Longleftrightarrow\ & 1 - y^2 < \sin^2(2^{-k}) \\
\Longrightarrow\ & \sqrt{1-y^2} < \sin(2^{-k}) && (\text{since } -1 < y < 0) \\
\Longleftrightarrow\ & \cos(\arcsin(y)) < \sin(2^{-k}) \\
\Longleftrightarrow\ & \sin(\frac{\pi}{2})\cos(\arcsin(y)) < \sin(2^{-k}) \\
\Longleftrightarrow\ & \sin(\frac{\pi}{2})\cos(\arcsin(y)) + \cos(\frac{\pi}{2})\sin(\arcsin(y)) < \sin(2^{-k}) \\
\Longleftrightarrow\ & \sin(\frac{\pi}{2} + \arcsin(y)) < \sin(2^{-k}) \\
\Longleftrightarrow\ & \frac{\pi}{2} + \arcsin(y) < 2^{-k} && (\sin \text{ is strictly increasing on } [0, \tfrac{\pi}{2}]) \\
\Longrightarrow\ & \left|-(\frac{\pi}{2} + \arcsin(y))\right| < 2^{-k} && (\arcsin(y) > -\tfrac{\pi}{2}) \\
\Longrightarrow\ & \left|-\frac{\pi}{2} - \arcsin(y)\right| < 2^{-k}
\end{aligned}$$

□

Now, we need to define a **WhileCC**-program approximating $\arcsin(f(x))$ for $-1 < f(x) < 1$ itself.

Definition 3.10.2. Let $F : \mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$, and and $x \in \mathbb{R}$. Then the **WhileCC**-procedure

```
pArcSine_F ≡ proc
                in $x$ : real $n$ : nat
                aux $i$ : nat $dist$ : real $res$ : real
             begin
                $res, i$ := choose ($y$ : real, $m$ : nat) :
                                 $\mathsf{Sine}(y) < F(x) < \mathsf{Sine}(y + 2^{-m})$
                return $res$
             end
```

of type $\mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ has the semantics

$$\mathsf{pArcSin_F}^{\mathcal{R}}(x,n) = \{y \in \mathbb{Q} \mid\ \sin(y) < f(x) < \sin(y + 2^{-n})\}$$

The **WhileCC**-procedure $\mathsf{pArcSine}_F$ outputs values sufficiently close to $\arcsin(f(x))$ only for $-1 < f(x) < 1$. The behaviour of $\mathsf{pArcSine}_F$ on $f(x) \in (-\infty, -1)$ or $f(x) \in (1, +\infty)$ is unknown. Now, incorporating the procedure isCloseEnough into the **WhileCC**-procedure, we get a procedure for **WhileCC**-approximating the totalized version $\arcsin'(f(x))$.

Definition 3.10.3. Let $F : \mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$, and and $x \in \mathbb{R}$. Then the **WhileCC**-procedure

```
ArcSin_F ≡ proc
             in $x$ : real $c$ : nat
             aux $chosenVal$ : nat $u$ : real $l$ : real $r$ : real
           begin
             $u$ := choose ($q$ : real) : $0 < q < 1$ and isCloseEnough($q, c + 1$)
             $l$ := choose ($q$ : real) : $-1 < q < 0$ and isCloseEnough($q, c + 1$)

             $chosenVal$ := choose ($k$ : nat) : proc
                                                   in $k$ : nat $x$ : real
                                                   aux $res$ : real
                                                 begin
                                                   if $k =_{nat} 1$ then
                                                     $res := -1 < F(x) < 1$
                                                   else if $k =_{nat} 2$ then
                                                     $res := F(x) > u$
                                                   else if $k =_{nat} 3$ then
                                                     $res := F(x) < l$
                                                   else
                                                     $res$ := ff
                                                   fi
                                                   return $res$
                                                 end
             if $chosenVal =_{nat} 1$ then
               $r$ := pArcSin$_F(x, c)$
             else if $chosenVal =_{nat} 2$ then
               $r$ := pArcSin$_F(u, c)$
             else if $chosenVal =_{nat} 3$ then
               $r$ := pArcSin$_F(l, c)$
             fi
             return $r$
           end
```

of type real $\times$ nat $\to$ real has the semantics

$$\begin{aligned}\mathsf{ArcSin}_F^{\mathcal{R}}(x,n) \overset{def}{=} \; & \{y \in \mathsf{pArcSin}_F^{\mathcal{R}}(x,n) \mid -1 < f(x) < 1\} \cup \\ & \{y \in \mathsf{pArcSin}_F^{\mathcal{R}}(u,n) \mid \mathsf{isCloseEnough}^{\mathcal{R}}(u,n) = \{\mathsf{tt}\} \\ & \quad \wedge \; 0 < u < 1 \wedge f(x) > u\} \cup \\ & \{y \in \mathsf{pArcSin}_F^{\mathcal{R}}(l,n) \mid \mathsf{isCloseEnough}^{\mathcal{R}}(l,n) = \{\mathsf{tt}\} \\ & \quad \wedge -1 < l < 0 \wedge f(x) < l\}\end{aligned}$$

Now, we need to prove that ArcSin$_F$ approximates $\arcsin(f(x))$.

Lemma 3.10.4. Let F : real $\times$ nat $\to$ real be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$, and $x \in \mathbf{dom}(f)$, then we have $\mathsf{ArcSin}_F^{\mathcal{R}}(x,n) \neq \emptyset$ for any $n \in \mathbb{N}$.

Proof. Let f be a **WhileCC**-approximable function, and $x \in \mathbf{dom}(f)$ and $n \in \mathbb{N}$. Then we have different possible cases based on the value of $f(x)$,

- $-1 < f(x) < 1$: In this case, it suffices to show that $\{\mathsf{pArcSin}_F^{\mathcal{R}}(x,n)\} \neq \emptyset$. Since the interval $(-1,1)$ is an open interval, $f(x)$ is an inner point, and sin is strictly increasing on $(-1,1)$, it means there is a rational y such that $-1 < \sin(y) < f(x) < \sin(y+2^n) < 1$, and hence $\mathsf{ArcSin}_F^{\mathcal{R}}(x,n) \neq \emptyset$.
- $f(x) \geq 1$: In this case, we need to show
$$\{y \in \mathsf{pArcSin}_F^{\mathcal{R}}(u,n) \mid \cos^2(2^{-n}) < u^2 \wedge 0 < u < 1 \wedge f(x) > u\} \neq \emptyset.$$
It suffices to show that there is some $u \in \mathbb{R}$ with $0 < u < 1 \leq f(x)$ such that $\cos^2(2^{-n}) < u^2$ and since $2^{-(n+1)} > 0$ meaning $\cos^2(2^{-n}) \neq 1$, there is a rational number $0 < u < 1$ such that $\cos^2(2^{-n}) < u^2 < 1$. Now that we found such u, by part 1, we can see that $\mathsf{pArcSin}_F^{\mathcal{R}}(u,n) \neq \emptyset$.
- $f(x) \leq -1$: We need to show
$$\{y \in \mathsf{pArcSin}_F^{\mathcal{R}}(l,n) \mid \cos^2(2^{-n}) < l^2 \wedge f(x) \leq -1 < l < 0 \wedge f(x) < l\} \neq \emptyset.$$
It suffices to show that there is some $l \in \mathbb{R}$ such that $\cos^2(2^{-n}) < l^2$ and also $-1 < l < 0$. since $2^{-n} > 0$, meaning $\cos^2(2^{-n}) \neq 1$, this means there is a rational numbers $-1 < l < 0$ such that $\cos^2(2^{-n}) < l^2 < 1$. Having found $-1 < l < 0$, by part 1, we can see that $\mathsf{pArcSin}_F^{\mathcal{R}}(l,n) \neq \emptyset$.

□

Lemma 3.10.5. Let F : $\mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$. For any $x \notin \mathbf{dom}(f)$, we have $\mathsf{ArcSin}_F^{\mathcal{R}}(x,n) = \emptyset$ for any $n \in \mathbb{N}$.

Proof. Let f be a **WhileCC**-approximable function, and $x \notin \mathbf{dom}(f)$ and $n \in \mathbb{N}$. Since $f(x)$ is not defined, based on the definition of the procedure and using Lemma 3.2.1, $F(x) < b$ would not terminate on any bound b. Hence we can conclude that $\mathsf{ArcSin}_F^{\mathcal{R}}(x,n) = \emptyset$. □

Lemma 3.10.6. Let F : $\mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$. We have $\forall x \in \mathbf{dom}(f)$, $\forall n \in \mathbb{N}$, $\forall y \in \mathsf{ArcSin}_F^{\mathcal{R}}(x,n)$:
$$|\arcsin'(f(x)) - y| < 2^{-n}.$$

Proof. For arbitrary $x \in \mathbf{dom}(f)$, $n \in \mathbb{N}$, and for any $y \in \mathsf{ArcSin}_F^{\mathcal{R}}(x,n)$, we have 3 cases to consider:

- $-1 < f(x) < 1$: In which case $y \in \mathsf{pArcSin}_F^{\mathcal{R}}(x,n)$. Then by definition of $\mathsf{pArcSin}_F$:
$$\begin{aligned} & -1 < f(x) < 1 \wedge \sin(y) < f(x) < \sin(y+2^{-n}) \\ \Longrightarrow\ & y < \arcsin(f(x)) < y + 2^{-n} \\ \Longrightarrow\ & \arcsin(f(x)) > y \ \wedge\ \arcsin(f(x)) - y < 2^{-n} \\ \Longrightarrow\ & |\arcsin(f(x)) - y| < 2^{-n} \\ \Longrightarrow\ & |\arcsin'(f(x)) - y| < 2^{-n}. \end{aligned}$$
- $f(x) > u$: for some $0 < u < 1$ in which case $y \in \{\ y \in \mathsf{pArcSin}_F^{\mathcal{R}}(u,n) \mid \cos^2(2^{-n}) < u^2\ \}$, hence:
$$\begin{aligned} & 1 - \sin^2(2^{-k}) < u^2 \wedge 0 < u < 1 \wedge f(x) > u \\ \Longrightarrow\ & \left|\frac{\pi}{2} - y\right| < 2^{-n} && \text{(by Lemma 3.10.1 part (1))} \\ \Longrightarrow\ & |\arcsin'(f(x)) - y| < 2^{-n} \end{aligned}$$

- $f(x) < l$ for some l satisfying $\cos^2(2^{-n}) < l^2 \wedge -1 < l < 0$: We would have $y \in \mathsf{pArcSin}^{\mathcal{R}}_F(l, n)$, in which case:

$$\begin{aligned}
& 1 - \sin^2(2^{-k}) < l^2 \wedge -1 < l < 0 \wedge f(x) > l \\
\implies & \left| -\frac{\pi}{2} - y \right| < 2^{-n} && \text{(by Lemma 3.10.1 part (2))} \\
\implies & |\arcsin'(f(x)) - y| < 2^{-n}
\end{aligned}$$

□

Theorem 3.10.7. Let F : real × nat → real be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$. Then the **WhileCC**-procedure $\mathsf{ArcSin}^{\mathcal{R}}_F$ approximates $\arcsin'(f(x))$.

Proof. Follows from Lemmas 3.10.4, 3.10.5, and 3.10.6. □

3.11 Natural Root

As mentioned in Definition 3.1.1, we modify the common definition of n-th root function $\mathrm{root}_{n,f}(x)$ for $n \in \mathbb{N}$ to a totalized version

$$\mathrm{root}_{n,f}(x) \overset{def}{=} \begin{cases} \sqrt[n]{f(x)} & \text{if } n \text{ is odd} \\ \sqrt[n]{f(x)} & \text{if } n \text{ is even and } f(x) > 0 \\ 0 & \text{if } n \text{ is even and } f(x) \le 0 \end{cases} \tag{3.5}$$

to obtain a continuous function with an open domain.
Now depending on the parity of n, different **WhileCC**-procedures are devised to approximate the function $\mathrm{root}_{n,f}(x)$.

3.11.1 Odd root

First of all, let us define a **WhileCC**-procedure that approximates $\sqrt[n]{f(x)}$ for odd ns.

Definition 3.11.1. Let F : real × nat → real be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$, and and $x \in \mathbb{R}$. Then the **WhileCC**-procedure

```
oddRoot$_{n,F}$ ≡ proc
                    in $x$ : real $n$ : nat
                    aux $res$ : real
                  begin
                    $res$ := choose $(y : \text{real}) : y^n < F(x) < (y + 2^{-m})^n$
                    return tt
                  end
```

of type real × nat → real has the semantics

$$\mathsf{oddRoot}^{\mathcal{R}}_{n,F}(x, m) = \{y \in \mathbb{Q} \mid \; (y^n < F(x))^{\mathcal{R}} = \{\mathsf{tt}\} \wedge (F(x) < (y + 2^{-m})^n)^{\mathcal{R}} = \{\mathsf{tt}\}\}$$

We prove that oddRoot approximates $\sqrt[n]{f(x)}$ for odd ns.

Lemma 3.11.2. Let F : real × nat → real be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$, and $x \in \mathbf{dom}(f)$, then for any odd $n \in \mathbb{N}$, we have $\mathsf{oddRoot}^{\mathcal{R}}_{n,F}(x, m) \neq \emptyset$ for any $n, m \in \mathbb{N}$.

Proof. We need to show that $\mathsf{oddRoot}^{\mathcal{R}}_{n,F}(x,m) \neq \emptyset$. Since n is odd, we can always choose a rational y such that $y^n < f(x) < (y + 2^{-m})^n$. Hence $\mathsf{oddRoot}_{n,F}(x,m) \neq \emptyset$. □

Lemma 3.11.3. Let F : $\mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$, and $x \notin \mathbf{dom}(f)$. Then we have $\mathsf{oddRoot}^{\mathcal{R}}_{n,F}(x,m) = \emptyset$ for any $n, m \in \mathbb{N}$.

Proof. Since $x \notin \mathbf{dom}(f)$, the comparison procedure $(- < F(-))$ with the second argument being x does not terminate and hence we can conclude that $\mathsf{oddRoot}^{\mathcal{R}}_{n,F}(x,m) = \emptyset$. □

Lemma 3.11.4. Let F : $\mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$. For any odd $n \in \mathbb{N}$, we have $\forall x \in \mathbf{dom}(f)$, $\forall m \in \mathbb{N}$, $\forall y \in \mathsf{oddRoot}^{\mathcal{R}}_{n,F}(x,m)$:

$$|\mathrm{root}_{n,f}(x) - y| < 2^{-n}.$$

Proof. For arbitrary $x \in \mathbf{dom}(f)$, odd $n \in \mathbb{N}$, and for any $y \in \mathsf{oddRoot}^{\mathcal{R}}_{n,F}(x,n)$, we have

$$\begin{aligned} & y^n < f(x) < (y + 2^{-m})^n \\ \implies & \left| \sqrt[n]{f(x)} - y \right| < 2^{-m} \end{aligned}$$

□

Theorem 3.11.5. Let F : $\mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$. Then $\mathsf{oddRoot}^{\mathcal{R}}_{n,F}$ **WhileCC**-approximates the function $\mathrm{root}_{n,f}(x)$ for any odd $n \in \mathbb{N}$.

Proof. Follows directly from Lemmas 3.11.2, 3.11.3, and 3.11.4. □

3.11.2 Even root

In this section, we construct a **WhileCC**-procedure that approximates $\sqrt[n]{f(x)}$ for even ns. Having the modifications in equation 3.5 in mind, the trick here is to find a bound for $f(x)$ such that, for any value less than this bound, approximating the n-th root of that value by outputting 0 is sufficiently accurate.

Definition 3.11.6. Let us define the **WhileCC**-procedure isCloseEnough of type $\mathsf{real} \times \mathsf{nat} \times \mathsf{nat} \to \mathsf{bool}$ below:

```
isCloseEnough ≡ proc
                  in $y$ : real $n$ : nat $k$ : nat
                begin
                  return $0 < y$ and $y < 1/2^{nk}$
                end
```

with the semantics

$$\mathsf{isCloseEnough}^{\mathcal{R}}(y,n,k) = \{\mathsf{tt} \mid (0 < y)^{\mathcal{R}} = \{\mathsf{tt}\} \wedge (y < 1/2^{nk})^{\mathcal{R}} = \{\mathsf{tt}\}\}$$

Lemma 3.11.7 (Bounds for Natural Root). For any even $n \in \mathbb{N}$, $y \in \mathbb{R}$ and $k \in \mathbb{N}$:

$$\mathsf{isCloseEnough}^{\mathcal{R}}(y,n,k) = \{\mathsf{tt}\} \implies |\sqrt[n]{y}| < 2^{-k}.$$

Proof. Let us assume $y \in \mathbb{R}$, $k \in \mathbb{N}$. Then,

$$\begin{aligned} & \mathsf{isCloseEnough}^{\mathcal{R}}(y,n,k) = \{\mathsf{tt}\} \\ \Longrightarrow\ & 0 < y < 2^{-nk} \\ \Longrightarrow\ & |y| < 2^{-nk} && \text{(since } 0 < y\text{)} \\ \Longrightarrow\ & |\sqrt[n]{y}| < 2^{-k} \end{aligned}$$

□

So if $\mathsf{isCloseEnough}^{\mathcal{R}}(y,n,k)$ outputs tt, we can use the fact that $\sqrt[n]{y}$ is at most 2^{-k} away from 0.

Remark 3.11.8. For any even n and fixed $k \in \mathbb{N}$, there is always an $y \in \mathbb{R}$ such that

$$\mathsf{isCloseEnough}^{\mathcal{R}}(y,n,k) = \{\mathsf{tt}\}.$$

Now let us define the **WhileCC**-procedure for approximating an even root of $f(x)$ for any **WhileCC**-approximable function $f : \mathbb{R} \to \mathbb{R}$.

Definition 3.11.9. Let F : real × nat → real be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$, and and $x \in \mathbb{R}$. Then the **WhileCC**-procedure

```
evenRoot_{n,F} ≡ proc
                   in n : nat x : real c : nat
                   aux chosenVal : nat l : real r : real
                 begin
                   l := choose (q : real) : isCloseEnough(q, c, n)

                   chosenVal := choose (k : nat) : proc
                                                     in k : nat x : real
                                                     aux res : real
                                                   begin
                                                     if k =_nat 1 then
                                                       res := 0 < F(x)
                                                     else if k =_nat 2 then
                                                       res := F(x) < l
                                                     else
                                                       res := ff
                                                     fi
                                                     return res
                                                   end
                   if chosenVal =_nat 1 then
                     r := oddRoot_{n,F}(x, c)
                   else if chosenVal =_nat 2 then
                     r := 0
                   fi
                   return r
                 end
```

of type $\mathsf{real} \times \mathsf{real} \to \mathsf{nat}$, has the semantics

$$\begin{aligned}\mathsf{evenRoot}^{\mathcal{R}}_{n,F}(x,m) = \{\ & y \in \mathsf{oddRoot}^{\mathcal{R}}_{n,F}(x,m) \mid (0 < F(x))^{\mathcal{R}} = \{\mathsf{tt}\}\} \cup \\ \{\ & 0 \mid \exists l \in \mathbb{Q} \quad \mathsf{isCloseEnough}(l,m,n) = \{\mathsf{tt}\} \wedge f(x) < l\}\end{aligned}$$

The procedure above reuses the **WhileCC**-procedure $\mathsf{oddRoot}$, originally defined for approximating odd roots, to approximate even roots since $\mathsf{oddRoot}$ happens to work for when n is even on $(0, +\infty)$. We now prove that $\mathsf{oddRoot}$ approximates $\sqrt[n]{f(x)}$ for even ns.

Lemma 3.11.10. Let $F : \mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$, n some even number, and $x \in \mathbf{dom}(f)$. Then we have $\mathsf{evenRoot}_{n,F}(x,m) \neq \emptyset$ for any $m \in \mathbb{N}$.

Proof. Let $F : \mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$, $x \in \mathbf{dom}(f)$ and $n \in \mathbb{N}$. We have two cases based on the value of $f(x)$:

- $0 < f(x)$: We need to show that $\mathsf{oddRoot}_{n,F}(x,m) \neq \emptyset$. Since the interval $(0, +\infty)$ is an open interval, $f(x)$ is an inner point, and the power function is strictly increasing on $(0, +\infty)$, we can choose a rational y such that $y^n < f(x) < (y + 2^{-m})^n$. Hence $\mathsf{oddRoot}_{n,F}(x,m) \neq \emptyset$.
- $f(x) \leq 0$: In this case, there is always a rational l such that

  $$\mathsf{isCloseEnough}(l,m,n) = \{\mathsf{tt}\}.$$

  This means that the set $\{\ 0 \mid \mathsf{isCloseEnough}(l,m,n) = \{\mathsf{tt}\} \wedge f(x) < l\}$ is non-empty, and so is $\mathsf{evenRoot}_{n,F}(x,m)$.

□

Lemma 3.11.11. Let $F : \mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$, and $x \notin \mathbf{dom}(f)$. Then we have $\mathsf{evenRoot}_{n,F}(x,m) = \emptyset$ for any $n, m \in \mathbb{N}$.

Proof. Since $x \notin \mathbf{dom}(f)$, the comparison procedure $(- < F(-))$ with the second argument being x does not terminate and hence we can conclude that $\mathsf{evenRoot}_{n,F}(x,m) = \emptyset$. □

Lemma 3.11.12. Let $F : \mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$. Then for any even n, we have $\forall x \in \mathbf{dom}(f)$, $\forall m \in \mathbb{N}$, $\forall y \in \mathsf{evenRoot}_{n,F}(x,m)$:

$$|\mathrm{root}_{n,f}(x) - y| < 2^{-n}.$$

Proof. We have two cases:

- $y \in \mathsf{oddRoot}^{\mathcal{R}}_{n,F}(x,m)$ and $(0 < F(x))^{\mathcal{R}} = \{\mathsf{tt}\}$: Then, by the definition of $\mathsf{oddRoot}_{n,F}(x,m)$:

  $$\begin{aligned} & y^n < f(x) < (y + 2^{-m})^n \\ \Longrightarrow & \left| \sqrt[n]{f(x)} - y \right| < 2^{-m}. \end{aligned}$$

- $y = 0 \wedge \exists l \in \mathbb{Q} \quad \mathsf{isCloseEnough}(l,m,n) = \{\mathsf{tt}\} \wedge f(x) < l$: Immediately follows from Lemma 3.11.7 that $\left| \sqrt[n]{l} \right| < 2^{-nm}$ and hence $|\mathrm{root}_{n,f}(x) - y| < 2^{-n}$.

□

Theorem 3.11.13. Let F : $\mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$. Then the **WhileCC**-procedure $\mathsf{evenRoot}_{n,F}(x, m)$ approximates $\mathrm{root}_{n,f}(x)$.

Proof. Follows directly from Lemmas 3.11.10, 3.11.11, and 3.11.12. □

3.12 Conclusion

Theorem 3.12.1. All elementary functions are **WhileCC**-approximable.

Proof. By induction on the structure of elementary functions, theorems 3.3.1 (constant function) and 3.3.2 (identity function) prove the base cases, and theorems

- 3.4.5 (addition),
- 3.5.6 (multiplication),
- 3.6.6 (division),
- 3.8.5 (logarithm),
- 3.7.12 (exponential),
- 3.9.10 (sin),
- 3.10.7 (arcsin), and
- 3.11.5, 3.11.13 (natural root)

prove the induction step. □

Chapter 4

Open Exhaustions for the Domains of Elementary Functions

This section is a first step towards proving that elementary functions are acceptable. In this section, we present an inductive construction for an effective open exhaustion for the domain of any arbitrary elementary function. To simplify the proofs, we will strengthen the claim and prove the following statement:

For any open set U with an effective open exhaustion, $f^{-1}(U)$ has an effective open exhaustion.

In particular, this stronger formulation makes the composition case trivial.

Definition 4.0.1 (Exhaustion-reflecting function). Let $f : \mathbb{R} \to \mathbb{R}$. We call f *exhaustion-reflecting* if for any open set U with an effective open exhaustion, $f^{-1}(U) \neq \emptyset$ implies that $f^{-1}(U)$ has an effective open exhaustion.

Remark 4.0.2. If f is exhaustion-reflecting, since $\mathbb{R}$ is an open set with an open exhaustion, $f^{-1}(\mathbb{R}) = \mathbf{dom}(f)$ has an effective exhaustion.

4.1 Preliminaries

In Section 4.1.1, we introduce some basic lemmas about effective open exhaustions. In Section 4.1.2 we generalize the concept of open exhaustions to $\mathbb{R}^n$. This generalization is used later in Sections 4.3 and 4.4.

4.1.1 Open Exhaustion Lemmas for $\mathbb{R}$

We begin by defining a simplified form of an effective open exhaustion called a "simple effective open exhaustion" (Definition 2.5.2). We prove that we can compute an effective open exhaustion from a simple effective open exhaustion. Simple effective open exhaustions are more convenient for the constructions in the remainder of Chapter 4.

Definition 4.1.1 (Simple effective open exhaustion). A sequence $(U_1, U_2, \ldots)$ of open sets in $\mathbb{R}$ is called a *simple effective open exhaustion* for an open set U if

- $U = \bigcup_{i=0}^{\infty} U_i$,
- for each $l \in \mathbb{N}$, U_l is a finite union of non-empty open finite intervals $I_1^l, I_2^l, ..., I_{k_l}^l$ whose closures are *not necessarily disjoint*, and

- (eventual covering property) for any $i, l \in \mathbb{N}$ there is some index $l' > l$ such that $\overline{U_l} = \bigcup_{i=0}^{k_l} \overline{I_i^l} \subseteq U_{l'}$ and also $U_l \subseteq U_{l+1}$,
- the map
$$l \mapsto (a_1^l, b_1^l, ..., a_{k_l}^l, b_{k_l}^l)$$
which delivers the sequence of endpoints of the stage $U_l = I_1^l \cup ... \cup I_{k_l}^l$ for any $l \in \mathbb{N}$, is recursive.

Remark 4.1.2. Note that the definition of "simple effective open exhaustion" is different from "effective open exhaustion" in three aspects:

- the closures of intervals in each stage of a simple effective open exhaustion are *not necessarily disjoint*,
- the closure of each stage of a simple effective open exhaustion is not necessarily contained in the next stage, instead it is replaced with the *eventual covering property*, and
- the map delivering the endpoints in each stage of a simple effective open exhaustion does not necessarily give us the intervals *in order*.

Lemma 4.1.3. There is a recursive function that given a simple open effective exhaustion for an open set, returns an effective open exhaustion for the same open set.

Proof. Let us take a simple effective open exhaustion for a set U. We give an algorithm to compute an effective open exhaustion for U. This algorithm consists of two steps:

1. merging and fixing the order
2. containing the closure in the immediate next stage

- **Step 1.** In any stage with index $l \in \mathbb{N}$, we first get all the finitely many open intervals and for overlapping intervals, we merge them into one open interval. Then we sort the intervals by their endpoints. This gives us the intervals
$$(a_1^l, b_1^l), \ldots, (a_{n_l}^l, b_{n_l}^l)$$
where n_l is the number of intervals we are left with, after merging all the intersecting ones. This sorting guarantees that
$$a_1^l < b_1^l \leq a_2^l < b_2^l \leq \cdots \leq a_{n_l-1}^l < b_{n_l-1}^l \leq a_{n_l}^l < b_{n_l}^l,$$
while preserving the eventual covering property.
We now need to make sure that the strict inequality
$$a_1^l < b_1^l < a_2^l < b_2^l < \cdots < a_{n_l-1}^l < b_{n_l-1}^l < a_{n_l}^l < b_{n_l}^l$$
holds, guaranteeing that the closure of these intervals are also disjoint. Now, taking any two intervals (a_k^l, b_k^l) and (a_{k+1}^l, b_{k+1}^l) that we have $a_k^l < b_k^l = a_{k+1}^l < b_{k+1}^l$, using the eventual covering property, this means the point b_k^l is covered in some later stage since it is contained in the closure of (a_k^l, b_k^l). This lets us merge any

such two intervals (a^l_k, b^l_k) and (a^l_{k+1}, b^l_{k+1}). This gives us a new set of intervals $(c^l_1, d^l_1), \ldots, (c^l_{m_l}, d^l_{m_l})$ with $m_l \leq n_l$ with

$$c^l_1 < d^l_1 < c^l_2 < d^l_2 < \cdots < c^l_{m_l-1} < d^l_{m_l-1} < c^l_{m_l} < d^l_{m_l}$$

which again preserves the eventual covering property. For the rest of this proof, we refer to the map generated after applying this step to each stage as the "transformed map".

- **Step 2.** Since the transformed map has the eventual covering property, this gives us a function $C : \mathbb{Q} \twoheadrightarrow \mathbb{N}$ that gives us, for any $q \in \mathbb{Q}$, the index of the first stage where q is covered. We define the map outputting the stages of the effective open exhaustion $(U_1, U_2, \ldots)$ of this algorithm inductively. For stage with index 1, we output the first stage of the transformed map. For stage with index l, let n_{l-1} be the number of intervals in the stage with index $l-1$ after the transformation. Our new stage with index l will be the transformed stage with index

$$\max\{C(c^l_1), C(d^l_1), \ldots, C(c^l_{n_{l-1}}), C(d^l_{n_{l-1}})\}.$$

□

Lemma 4.1.4 (Effective open exhaustion for intersection). Let U and V be open sets in $\mathbb{R}$ with effective open exhaustions such that $U \cap V \neq \emptyset$. Then $U \cap V$ has an effective open exhaustion.

Proof. Let $(U_1, U_2, \ldots)$ and $(V_1, V_2, \ldots)$ be effective open exhaustions for the sets U and V, respectively. Since by assumption we have $U \cap V \neq \emptyset$, we know there is some $m \in U \cap V$. We also know that both exhaustions $(U_1, U_2, \ldots)$ and $(V_1, V_2, \ldots)$ will cover m at some stage with index l_u resp. l_v. Note that we can effectively find l_u resp. l_v by looking through all the stages of the two exhaustions until a pair of intersecting intervals is found. Now let us take $k := \max\{l_u, l_v\}$. Then

$$(U_k \cap V_k, U_{k+1} \cap V_{k+1}, U_{k+2} \cap V_{k+2}, \ldots)$$

is an effective open exhaustion for $U \cap V$. □

Proposition 4.1.5. Let $U, V \subseteq \mathbb{R}$. Then if U and V have effective open exhaustions, then $U \cup V$ has an effective open exhaustion.

Proof. Let $U, V \subseteq \mathbb{R}$ and $(U_1, U_2, \ldots)$ and $(V_1, V_2, \ldots)$ be effective open exhaustions for the sets U and V, respectively. Then to prove $U \cup V$ has an effective open exhaustion, by Lemma 4.1.3, it suffices to give a simple effective open exhaustion. The sequence

$$(U_0 \cup V_0, \ldots, U_k \cup V_k, \ldots)$$

is a simple effective open exhaustion for $U \cup V$. □

Proposition 4.1.6 (Effective open exhaustion for a finite interval). The open interval (a, b) with $a, b \in \mathbb{Q}$ and $a < b$, has an effective open exhaustion

$$\left(\left(a + \frac{b-a}{3}, b - \frac{b-a}{3}\right), \ldots, \left(a + \frac{b-a}{n+3}, b + \frac{b-a}{n+3}\right), \ldots\right).$$

Proposition 4.1.7 (Effective open exhaustion for an open interval). The open interval $(a, +\infty)$ with $a \in \mathbb{Q}$ has an effective open exhaustion

$$\left((a+1,1), \left(a+\frac{1}{2}, 2\right), ..., \left(a+\frac{1}{k+1}, k+1\right), ...\right).$$

Lemma 4.1.8 (Effective open exhaustion with removing one point). Let $U \subseteq \mathbb{R}$ be an open set with an effective open exhaustion. Then for any $r \in \mathbb{Q}$, $U \setminus \{r\}$ has an effective open exhaustion.

Proof. Let $(U_1, U_2, \ldots)$ be an effective open exhaustion for U. Then consider the sequence

$$\left(U_0 \setminus [r-1, r+1], \ldots, U_k \setminus \left[r - \frac{1}{k+1}, r + \frac{1}{k+1}\right], \ldots\right).$$

There must be a stage m at which $U_m \setminus \left[r - \frac{1}{m}, r + \frac{1}{m}\right]$ is non-empty. Then the sequence

$$\left(U_m \setminus \left[r - \frac{1}{m+1}, r + \frac{1}{m+1}\right], \ldots\right).$$

is clearly an effective open exhaustion that for $U \setminus \{r\}$. □

Theorem 4.1.9 (Open exhaustion of intervals using WhileCC-approximability). Let F : $\mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ be a **WhileCC**-procedure approximating $f : \mathbb{R} \to \mathbb{R}$, and f be strictly monotone on the interval $[a, b] \subseteq \mathbf{dom}(f)$ with $a, b \in \mathbb{Q}$. Then there is an effective open exhaustion for $f((a, b))$.

Proof. Here we discuss a strictly increasing function f. The case of a strictly decreasing function follows a similar logic. We need to define intervals $U_l = (x_l, y_l)$ with endpoints in $\mathbb{Q}$ with the following properties:

- $f(a) < x_l < y_l < f(b)$
- $\overline{(x_l, y_l)} \subseteq (x_{l+1}, y_{l+1})$
- $\bigcup_{k=1}^{\infty} (x_k, y_k) = (f(a), f(b))$

We present an algorithm to compute the mapping *intervals* : $\mathbb{N} \times \mathbb{N} \to \mathbb{I}^*$ delivering intervals in a stage. We begin by giving an informal description of the algorithm:

1. Start counters $n = 0$, $l = 0$.
2. Increase n until you get an approximation $x \in F^{\mathcal{R}}(a, n)$ and $y \in F^{\mathcal{R}}(b, n)$ with $x + 2^{-n} < y - 2^{-n}$.
3. Store x, y respectively in x_0, y_0.
4. Increase n until new approximations $x \in F^{\mathcal{R}}(a, n)$ and $y \in F^{\mathcal{R}}(b, n)$ are calculated with $x + 2^{-n} < x_l$ and $y_l < y - 2^{-n}$.
5. Increase l by one.
6. Store $x + 2^{-n}, y - 2^{-n}$ in x_l, y_l respectively and go to step 4.

The construction guarantees the three conditions above and equivalently outputting endpoints for one interval for each stage of the effective exhaustion for $(f(a), f(b))$. The alorithm we just defined computes the following function:

$$\begin{aligned}
\mathit{intervals}(l, 1) &= (x, y) \text{ such that } \exists n \in \mathbb{N}\ x \in F^{\mathcal{R}}(a, n) \wedge y \in F^{\mathcal{R}}(b, n) \\
&\quad \wedge x + 2^{-n} < y - 2^{-n} \\
\mathit{intervals}(l, i+1) &= (x + 2^{-n}, y - 2^{-n}) \text{ such that } \exists n \in \mathbb{N}\ x \in F^{\mathcal{R}}(a, n) \wedge y \in F^{\mathcal{R}}(b, n) \\
&\quad \wedge x + 2^{-n} < \mathrm{fst}(\mathit{intervals}(l, i)) \wedge y - 2^{-n} > \mathrm{snd}(\mathit{intervals}(j, i))
\end{aligned}$$

where fst returns the left element in a pair and snd returns the right element in the pair. □

Remark 4.1.10. Theorem 4.1.9 is used to prove the exhaustion-reflecting property (Definition 4.0.1) of the basic elementary functions (see Section 4.2). Note that Theorem 4.1.9 requires the closed interval $[a, b]$ to be in the domain of f. We can strengthen Theorem 4.1.9 to only require the open interval $(a, b) \subseteq \mathbf{dom}(f)$, however, the proof for the strengthened version is more complicated, and the current version suffices for our purposes.

4.1.2 Open Exhaustions in $\mathbb{R}^n$

In this section we generalize the notion of an effective open exhaustion to that of $\mathbb{R}^n$ by generalizing intervals in $\mathbb{R}$ to cubes in $\mathbb{R}^n$.

Definition 4.1.11 (Open n-cube). Let $I_1, \ldots, I_n \subseteq \mathbb{R}$ be open intervals. Then we call the open set $I_1 \times \cdots \times I_n \subseteq \mathbb{R}^n$ an *open n-cube.*

Definition 4.1.12 (Closed n-cube). Let $I_1, \ldots, I_n \subseteq \mathbb{R}$ be closed intervals. Then we call the closed set $I_1 \times \cdots \times I_n \subseteq \mathbb{R}^n$ a *closed n-cube.*

Definition 4.1.13 (Rational n-cube). Let $I_1, \ldots, I_n \subseteq \mathbb{R}$ be open (resp. closed) intervals. We call the open (resp. closed) set $I_1 \times \cdots \times I_n \subseteq \mathbb{R}^n$ a *rational open n-cube* (resp. rational closed n-cube) if all the endpoints of $I_1, \ldots, I_n$ are rational. The set $\mathbb{I}^n$ denotes the set of all rational n-cubes.

Definition 4.1.14 (Realizability for $\mathbb{I}^n$). We define the realizability relation $\Vdash_{\mathbb{I}^n}$ as the smallest relation satisfying

$$\frac{c_1 \Vdash_{\mathbb{Q}} a_1 \qquad c_2 \Vdash_{\mathbb{Q}} b_1 \qquad \cdots \qquad c_{2n-1} \Vdash_{\mathbb{Q}} a_n \qquad c_{2n} \Vdash_{\mathbb{Q}} b_n}{p_1^{a_1} p_2^{b_1} \cdots p_{2n-1}^{a_n} p_{2n}^{b_n} \Vdash_{\mathbb{I}^n} (a_1, b_1) \times \cdots \times (a_n, b_n)}$$

where p_i is the ith prime number.

Defining a realizability relation for $\mathbb{I}^n$ along with Definitions 2.3.7,2.3.4 and Definition 2.3.3 lets us define computable functions from $\mathbb{N}$ to finite sequences of cubes.

Definition 4.1.15 (Generalized open exhaustion). Let U be an open subset of $\mathbb{R}^n$, and $X = (U_0, U_1, U_2, \ldots)$ a sequence of open subsets of $\mathbb{R}^n$. The sequence X is called an *open exhaustion of U* iff

1. $U = \bigcup_{i=0}^{\infty} U_i$, and

2. for each $l \in \mathbb{N}$, U_l is a (not necessarily disjoint) finite union of non-empty open n-cubes $Q_1^l, Q_2^l, ..., Q_{k_l}^l$
3. for any $l \in \mathbb{N}$ there is some $j > l \in \mathbb{N}$ such that $\overline{U_l} = \bigcup_{i=0}^{k_l} \overline{Q_i^l} \subseteq U_j$, and also $U_l \subseteq U_{l+1}$.

For each $l \in \mathbb{N}$, U_l is called a *stage* of the exhaustion, with components $Q_1^l, Q_2^l, ..., Q_{k_l}^l$.

Definition 4.1.16 (Generalized effective open exhaustion). An open exhaustion $(U_1, U_2, \ldots)$ of an open set $U \subseteq \mathbb{R}^n$ is called *effective* if

- each stage U_l consists of finitely many open rational n-cubes $Q_1^l, \ldots, Q_{k_l}^l$, and
- the map
$$l \mapsto (Q_1^l, \ldots, Q_{k_l}^l)$$
which delivers the sequence of stages
$$U_l = (Q_1^l, \ldots, Q_{k_l}^l)$$
is recursive.

Remark 4.1.17. Note that the generalized notion of an effective open exhaustion in $\mathbb{R}^n$ given in Definition 4.1.16 for the case where $n = 1$, coincides with the concept of a simple effective open exhaustion in $\mathbb{R}$ given earlier in Definition 4.1.1

Definition 4.1.18 (Effective sequence of open exhaustions). Let $(U_0, U_1, \ldots)$ be a sequence of open sets in $\mathbb{R}^n$ each with an effective open exhaustion. We call the sequence $U_0, U_1, \ldots$ an *effective sequence of open exhaustions* if we have a recursive function

$$\mathit{Stage} : \mathbb{N} \times \mathbb{N} \to (\mathbb{I}^n)^*$$

taking in the index i of the open set, the stage s, and outputting the n-cubes in stage s of the effective open exhaustion of O_i.

Theorem 4.1.19 (Union of effective open exhaustions). Let $(U_0, U_1, \ldots)$ be an effective sequence of open exhaustions. Then $\bigcup_{i \in \mathbb{N}} U_i$ has an effective open exhaustion.

Proof. Let us assume the effective the open exhaustion for $U_0, U_1, \ldots$ is given by $(O_0), (O_1), \ldots$ and let us denote the lth stage of (O_i) by O_i^l (for $i \in \mathbb{N}$).

Stage t of the effective open exhaustion for $\bigcup_{i \in \mathbb{N}} U_i$ consists of the union of the first t stages of the first t open exhaustions. Let us construct the effective open exhaustion (S_t) formally by

$$S_t = \bigcup_{k=0}^{t-1} O_k^t.$$

We can clearly compute the above open exhaustion for $\bigcup_{i \in \mathbb{N}} U_i$ since each O_k^t is a finite union of finitely many intervals. □

Corollary 4.1.20. Let $f : \mathbb{R}^n \to \mathbb{R}^m$. Let us assume for any rational open m-cube Q that $f^{-1}(Q)$ has an effective open exhaustion (O_l^Q). If the function

$$\mathit{Func} : \mathbb{I}^m \times \mathbb{N} \to (\mathbb{I}^n)^*$$

that gets an m-cube Q and a stage index l and outputs the encoding of the sequence of intervals in the lth stage of $f^{-1}(Q)$ is recursive, then for any open set U with an effective open exhaustion, $f^{-1}(U)$ has an effective open exhaustion.

Remark 4.1.21. This means to prove that a function $f : \mathbb{R}^n \to \mathbb{R}^m$ is exhaustion-reflecting, it suffices to show that we can compute, for any rational open m-cube Q, an open exhaustion for $f^{-1}(Q)$.

Theorem 4.1.22. Let $U \subseteq \mathbb{R}^2$ be an open set. If there is a recursive function

$$\mathrm{in}_U : \mathbb{Q}^2 \times \mathbb{Q}^2 \to \mathbb{B}$$

taking (a_1, b_1, a_2, b_2) as inputs and deciding whether the closed rational 2-cube

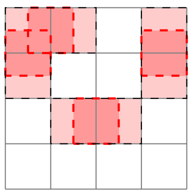
$$[a_1, b_1] \times [a_2, b_2]$$

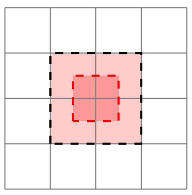
is completely contained in U, then U has an effective open exhaustion.

Proof. We give an algorithm that constructs an effective open exhaustion. Intuitively, to generate each stage of the open exhaustion, we start by considering a grid and selecting every open square Q in the grid such that $\overline{Q} \subset U$. Note that U can be unbounded which implies that we can have infinite number of squares. We work around this by selecting squares that lie within an increasingly larger *search radius* around the origin. If the grid is too coarse and no grid square (and its closure) falls fully under U, we keep dividing each side of the square into two (resulting in the division of each square to four squares) until there is a square whose closure falls under U. Such a square must exist since U is a non-empty open set and the rational numbers are dense in $\mathbb{R}$.

The selected squares will then be investigated for adjacency: for each two selected squares that are adjacent vertically or horizontally, a filler open square (as shown in Figure 4.1a) will be added. In addition, if four squares are adjacent (as shown in Figure 4.1b), an additional filler covering the center will be added. Note that in figure 4.1b, the four two-square-fillers added in the previous step are not drawn for clarity. The set of selected open squares and added open fillers form a stage of the open exhaustion.

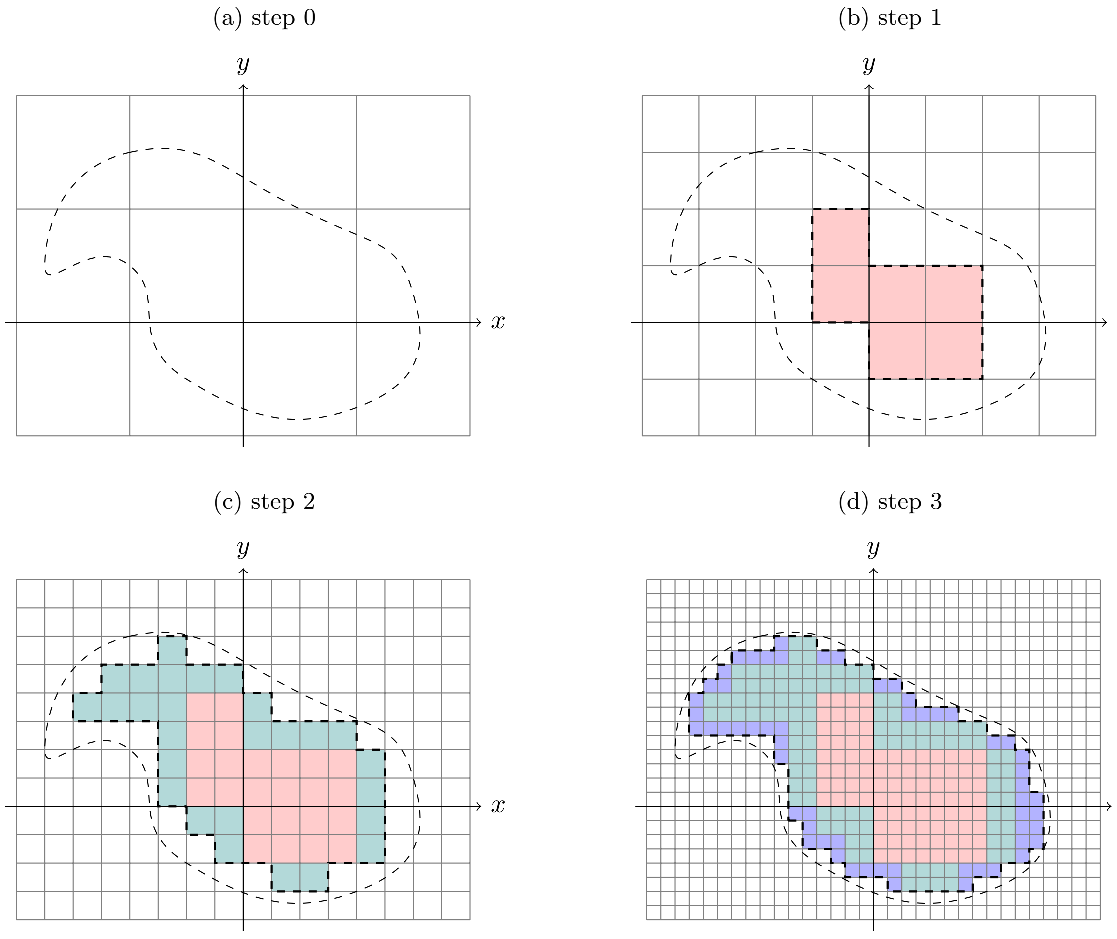
Figure 4.1: Visualization of the additional fillers


(a) two neighbouring squares


(b) four neighbouring squares

To get to the next stage, we keep the squares for the current stage and subdivide the grid further. Then we repeat the process of checking if any remaining uncovered squares (and their closures) fall under U completely, and check if any filling is needed. Using this method, it is guaranteed that each point in U will eventually be covered by these open squares. A visual trace for this algorithm on an open set is shown in Figure 4.2. Note that in this example, the search radius covers U entirely from step 0.

Figure 4.2: Visualization of the algorithm outputting an open exhaustion using a decision procedure



Note that here the filler squares are not shown, and only the area that is covered at each stage is shown. Step 0 is when the algorithm starts. The output for stage 0 (resp. 1 and 2) is shown in step 1 (resp. 2 and 3). Different colors show in which stage the points are covered using the open exhaustion.

□

4.2 Basic Functions

In this section, we show that all the basic elementary functions are exhaustion-reflecting (Definition 4.0.1). This implies that the domain of basic elementary functions have effective open exhaustions which constitutes the first condition for acceptability.

Theorem 4.2.1. Let $f(x) = c$ for any computable real number c. Then f is exhaustion-reflecting.

Proof. Let U be an open set with an open exhaustion $(U_1, U_2, \ldots)$ and $f^{-1}(U) \neq \emptyset$. This means we have $f^{-1}(U) = \mathbb{R}$, and the sequence $(O_1, O_2, \ldots)$ defined by $O_k = (-k-1, k+1)$, is an effective open exhaustion for $\mathbb{R}$. □

Theorem 4.2.2. The identity function $\mathrm{id}(x) = x$ is exhaustion-reflecting.

Proof. Let U be an open set with an open exhaustion X. By definition $\mathrm{id}^{-1}(U) = U$, and hence X is clearly an effective open exhaustion for $\mathrm{id}^{-1}(U)$ as well. $\square$

Theorem 4.2.3. Let $\mathrm{inv}(x) = \frac{1}{x}$. Then $\mathrm{inv}(x)$ is exhaustion-reflecting.

Proof. Using Remark 4.1.21, it suffices to show that, for any non-empty interval $I = (a, b) \subseteq \mathbb{R}$ with $a, b \in \mathbb{Q}$, we have an open exhaustion for $\mathrm{inv}^{-1}(I)$. We prove this by cases:

- $a = 0$: Proposition 4.1.6 gives us an effective open exhaustion for $\mathrm{inv}^{-1}(I) = (0, \frac{1}{b})$.
- $b = 0$: Proposition 4.1.6 gives us an effective open exhaustion for $\mathrm{inv}^{-1}(I) = (\frac{1}{a}, 0)$.
- $a < 0$ and $b > 0$: Proposition 4.1.6 gives us an effective open exhaustion for $\mathrm{inv}^{-1}(I) = (\frac{1}{a}, 0) \cup (0, \frac{1}{b})$. Then, we can construct an effective open exhaustion for the union of the two sets using Proposition 4.1.5.
- $a, b < 0$ or $a, b > 0$: Proposition 4.1.6 gives us effective open exhaustions for $\mathrm{inv}^{-1}(I) = (\frac{1}{b}, \frac{1}{a})$.

$\square$

Theorem 4.2.4. Let $f(x) = root_{n,id}(x)$ as defined in subsection 3.11. Then f is exhaustion-reflecting.

Proof. Let us assume U is an open set with an open exhaustion X. Now, let us consider the parity of n:

- Case of odd n: Since we have the effective open exhaustion X, we can go through each stage, and compute the corresponding stage.
  At stage k, we go through the endpoints of intervals, and for each (a_i, b_i) we encounter, we write (a_i^n, b_i^n).
- Case of even n: Since we have the effective open exhaustion X, again we can go through each stage. At each stage, we go through the endpoints of the intervals in that stage, and for each (a_i, b_i) we encounter, we have three possibilities:
  - Case a_i, b_i are both positive: we write (a_i^n, b_i^n).
  - Case a_i, b_i are both negative: we ignore the interval since this interval does not fall under the image of $root_{n,id}$ for our even n.
  - Case a_i is negative, but b_i is positive: here we need to accommodate the modifications we made in section 3.11, so we write $(-k, b_i^n)$ so that all negative numbers are eventually covered.

$\square$

Theorem 4.2.5. The function $\ln(x)$ is exhaustion-reflecting.

Proof. Let us take any arbitrary open set U with an effective open exhaustion. We want to come up with an effective open exhaustion for $\ln^{-1}(U) = \exp(U)$.

Using Remark 4.1.21, we only need to prove that the preimage of any open interval $I = (a, b)$ with $a, b \in \mathbb{Q}$ has an effective open exhaustion.

The pre-image function for ln is exp.

We know that

- The function exp is monotonically increasing.
- The function exp is **WhileCC**-approximable (Using Theorem 3.7.12).
- The function exp is defined on any arbitrary interval (a, b).

Hence, Theorem 4.1.9 gives us an open exhaustion for $\exp(I)$, and this completes the proof. $\square$

Theorem 4.2.6. The function $\exp(x) = e^x$ is exhaustion-reflecting.

Proof. Let us take any arbitrary open set U with an effective open exhaustion. Since $\exp(x) > 0$, we know that $\exp^{-1}(U) = \exp^{-1}(U \cap (0, +\infty))$. We want to come up with an effective open exhaustion for $\exp^{-1}(U \cap (0, +\infty))$. Using Remark 4.1.21, we only need to prove that the preimage of any open interval $I = (a, b)$ with $a, b \in \mathbb{Q}$ has an effective open exhaustion. The pre-image function for exp is ln. Since ln is only defined on positive reals, $\ln(I) = \ln((a, b) \cap (0, +\infty))$.
We know that

- The function ln is monotonically increasing.
- The function ln is **WhileCC**-approximable (using Theorem 3.8.5).
- The function ln is defined on $(a, b) \cap (0, +\infty)$.

Hence, Theorem 4.1.9 gives us an open exhaustion for $ln(I)$, and this completes the proof. $\square$

Lemma 4.2.7. Let $f(x) = \sin(\pi x)$ and $I = (a, b)$ with $a, b \in \mathbb{Q}$ and $-1 < a < b < 1$. Then there is an effective open exhaustion for $f^{-1}(I)$.

Proof. Let us consider $f^{-1}(x) = \arcsin(x)/\pi$. The function f^{-1} is clearly **WhileCC**-approximable. We know that

- The function f^{-1} is monotonically increasing.
- The function f^{-1} is defined on (a, b).

Hence, Theorem 4.1.9 gives us an open exhaustion for $f^{-1}(I)$, and this completes the proof. $\square$

Theorem 4.2.8. The function $\sin(x)$ is exhaustion-reflecting.

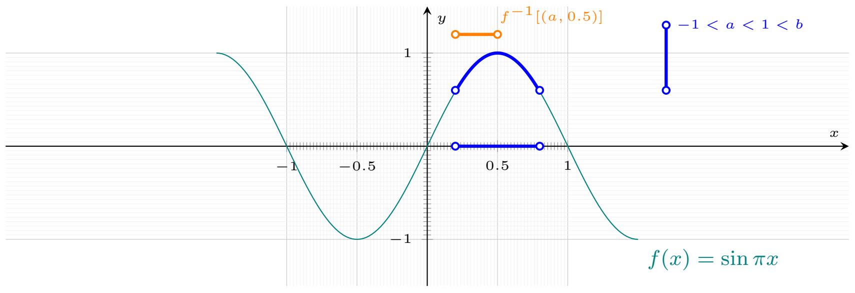
Proof. It suffices to show that $f(x) = \sin(\pi x)$ is exhaustion-reflecting. Using Remark 4.1.21, we only need to prove that $f^{-1}(I)$, if nonempty, has an effective open exhaustion for any $I = (a, b)$ with $a < b \in \mathbb{Q}$. Let us consider the value of a and b:

- $-1 < a < 1 < b$: In this case we can use Theorem 4.1.21 along with Lemma 4.2.7, we can get an open exhaustion for $f^{-1}((a, 1/2))$. We modify each (a_i, b_i) to

$$\begin{gathered}(a_i - 2i, -1 - a_i - 2i) \cup \cdots \cup (a_i - 2, -1 - a_i - 2) \\ \cup (a_i, -1 - a_i) \\ \cup (a_i + 2, -1 - a_i + 2) \cdots (a_i + 2i, -1 - a_i + 2i)\end{gathered}$$

Note that the reason we are modifying each interval is that the open exhaustion $f^{-1}((a, 1/2))$ does not cover $x = 1/2$, so stretching (a_i, b_i) to $(a_i, 1 - a_i)$ will help cover the point $x = 1/2$ as well as the mirrored interval $(1/2, 1 - f(a))$.

Figure 4.3: Covering the points with $f(x) = 1$



The reason we are adding $(a_i - 2i, -1 - a_i - 2i) \cup \cdots \cup (a_i - 2, -1 - a_i - 2)$ and $(a_i + 2, -1 - a_i + 2) \cdots (a_i + 2i, -1 - a_i + 2i)$ is that $f(x)$ is a periodic function and we would want to eventually cover all the xs in $\mathbb{R}$ for which $f(x)$ falls in $(a, 1]$. The intuition for building up the stages is shown in Figure 4.3.

- $-1 < a < b < 1$: In this case we can use Theorem 4.1.21 we can get an open exhaustion for $f^{-1}(I)$. For each stage we modify each (a_i, b_i) to

$$(a_i - 2i, b_i - 2i) \cup \cdots \cup (a_i, b_i) \cup \cdots \cup (a_i + 2i, b_i + 2i)$$

- $a, b > 1$ or $a, b < -1$: In this case $f^{-1}(I) = \emptyset$.
- $a < -1 < b < 1$: In this case we can use Theorem 4.1.21 we can get an open exhaustion for $f^{-1}((-1/2, b))$. We modify each (a_i, b_i) to

$$\begin{gathered}(-1 - b_i - 2i, b_i - 2i) \cup \cdots \cup (-1 - b_i - 2, b_i - 2) \\ \cup(-1 - b_i, b_i) \\ \cup(-1 - b_i + 2, b_i + 2) \cdots (-1 - b_i + 2i, b_i + 2i)\end{gathered}$$

- $a < -1 < 1 < b$: In this case the open exhaustion is

$$((-1, 1), \ldots, (-k - 1, k + 1), \ldots).$$

This gives us an effective open exhaustion for $f^{-1}(U)$ for any open set U.

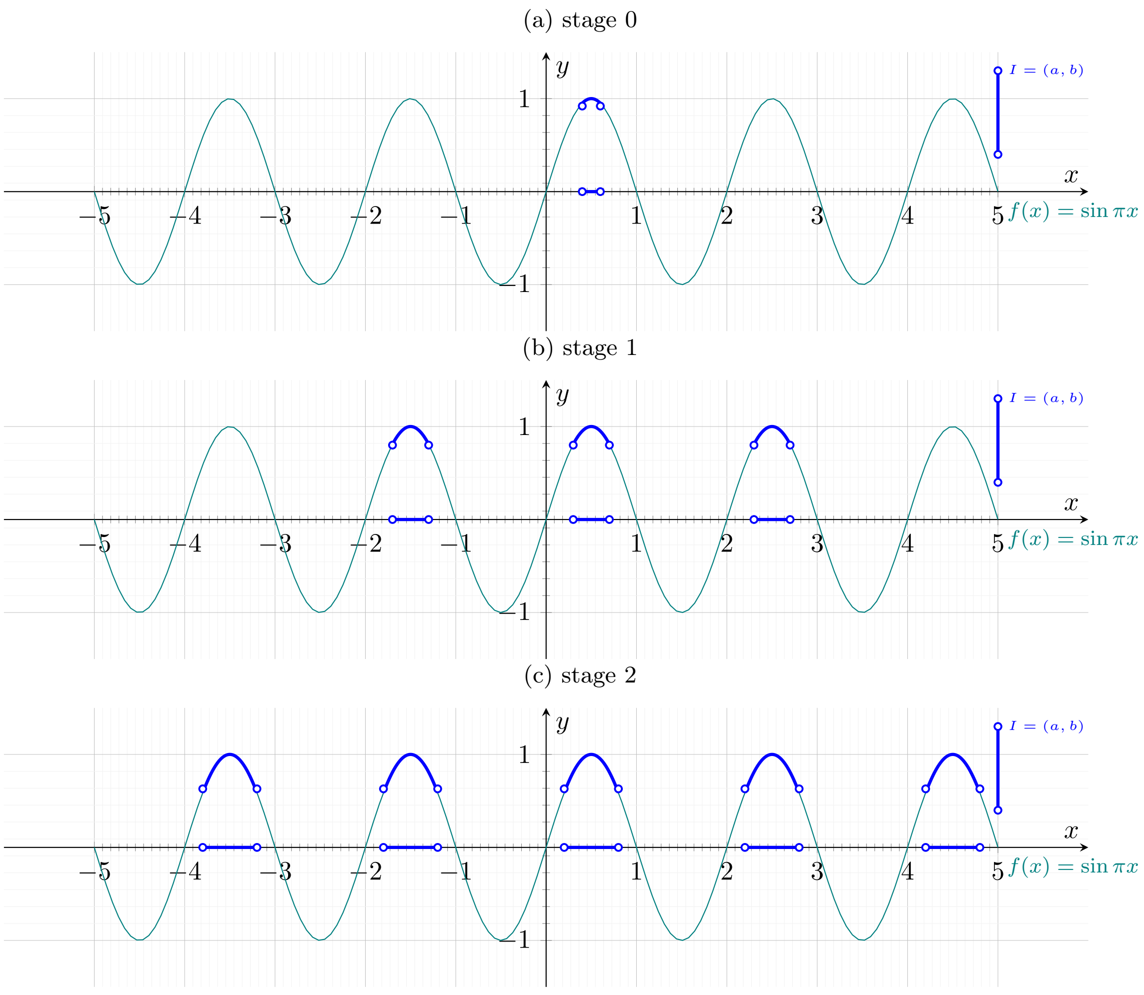
□

Theorem 4.2.9. The function $\arcsin'$ is exhaustion-reflecting.

Proof. It suffices to show that $f(x) = \arcsin'(x)/\pi$ is exhaustion-reflecting. Using Remark 4.1.21, we only need to prove that if $f^{-1}(I)$ has an effective open exhaustion for any $I = (a, b)$ with $a < b \in \mathbb{Q}$. Let us consider the value of a and b:

- $-1/2 < a < b < 1/2$: The pre-image function for $\arcsin'$ is sin. So we just need to construct an open exhaustion for $\sin(I)$. We know that
  - The function sin is monotonically increasing on (a, b).

Figure 4.4: Visualization of our algorithm for building up stages for the preimage of an interval (a, b) with $-1 < a < 1 < b$



   - The function sin is **WhileCC**-approximable (using Theorem 3.8.5).
   - The function sin is defined on (a, b).

  Hence, Theorem 4.1.9 gives us an open exhaustion for $\sin(I)$.

- $a, b > 1/2$ or $a, b < -1/2$: Since $\arcsin'(x) \in [-\frac{\pi}{2}, \frac{\pi}{2}]$, we know that

$$(\arcsin')^{-1}(U) = (\arcsin')^{-1}(U \cap [-\frac{\pi}{2}, \frac{\pi}{2}]).$$

  So for such $I = (a, b)$, $(\arcsin')^{-1}(I) = \emptyset$.

- $-1/2 < a < 1/2 < b$: We can use the Theorem 4.1.9 to get an open exhaustion for $\sin(I)$, but then at each stage, modify all the intervals (a_i, b_i) to $(a_i, i + 1)$ to cover $(1, +\infty)$.

- $a < -1/2 < b < 1/2$: We can use the Theorem 4.1.9 to get an open exhaustion for $\sin(I)$, but then at each stage, modify all the intervals (a_i, b_i) to $(-i - 1, b_i)$ to cover $(-\infty, -1)$.

- $a < -1/2$ and $1/2 < b$: In this case, the open exhaustion is

$$((-1, 1), \dots, (-k - 1, k + 1), \dots).$$

This gives us an effective open exhaustion for $(\arcsin')^{-1}(U)$. □

4.3 Composition of Functions

In this section, we prove that the composition of two exhaustion-reflecting functions is exhaustion-reflecting.

Theorem 4.3.1. Let $f : \mathbb{R}^n \to \mathbb{R}^k$ and $g : \mathbb{R}^m \to \mathbb{R}^n$ be exhaustion-reflecting, then $f \circ g : \mathbb{R}^m \to \mathbb{R}^k$ is also exhaustion-reflecting.

Proof. Let $U \subseteq \mathbb{R}^k$ be an open set with an effective open exhaustion. Then by assumption on f, $f^{-1}(U) \subseteq \mathbb{R}^n$ has an effective open exhaustion, and hence $g^{-1}(f^{-1}(U)) \subseteq \mathbb{R}^m$ has an effective open exhaustion. □

4.4 Addition and Multiplication

In this section we prove that the addition and multiplication of two exhaustion-reflecting functions $f, g : \mathbb{R} \to \mathbb{R}$, defined as $(f + g)(x) = f(x) + g(x)$ and $(f \dots g)(x) = f(x) \cdot g(x)$ respectively, are exhaustion-reflecting. In order to do this, we first break down the addition and multiplication into atomic functions (that when composed together give us our original addition and multiplication) and prove that each of these atomic functions are exhaustion-reflecting. Then using Theorem 4.3.1, we can immediately imply that our original addition and multiplication are exhaustion-reflecting.

4.4.1 Deconstruction of Addition and Multiplication

In order to prove properties about addition ($(f+g)(x) = f(x) + g(x)$) we deconstruct $(f+g)(x)$ into the composition of

- addition: $\mathrm{Add}(x, y) = x + y$,
- cartesian product: $(f \times g)(x, y) = (f(x), g(y))$,
- diagonal: $\mathrm{Diag}(x) = (x, x)$.

This gives us

$$\begin{aligned}\mathrm{Add}((f \times g)(\mathrm{Diag}(x))) &= (\mathrm{Add}((f \times g)(x, x))) \\ &= (\mathrm{Add}(f(x), g(x))) \\ &= f(x) + g(x)\end{aligned}$$

Similarly, in order to prove properties about multiplication ($(f \cdot g)(x) = f(x) \cdot g(x)$) we deconstruct $(f \cdot g)(x)$ into the composition of

- multiplication: $\mathrm{Mult}(x, y) = x \cdot y$,
- cartesian product: $(f \times g)(x, y) = (f(x), g(y))$,
- diagonal: $\mathrm{Diag}(x) = (x, x)$.

This gives us

$$\begin{aligned}\mathrm{Mult}((f \times g)(\mathrm{Diag}(x))) &= (\mathrm{Mult}((f \times g)(x, x))) \\ &= (\mathrm{Mult}(f(x), g(x))) \\ &= f(x) \cdot g(x)\end{aligned}$$

4.4.2 Exhaustion-reflection for addition

Lemma 4.4.1. The function $\mathrm{Add} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ with $\mathrm{Add}(x, y) = x + y$ is exhaustion-reflecting.

Proof. Using Remark 4.1.21, we only need to prove that for any interval $I = (a, b)$ with rational endpoints, we have an effective open exhaustion for $f^{-1}(I)$, i.e.,

$$\mathrm{Add}^{-1}(I) = \{(x, y) \mid x + y \in (a, b)\}$$

has an effective open exhaustion. Now using Theorem 4.1.22 we only need to present a recursive function deciding if any arbitrary 2-cube $(x_1, x_2) \times (y_1, y_2)$ is in $\mathrm{Add}^{-1}[I]$. As shown in Figure 4.5, we can define

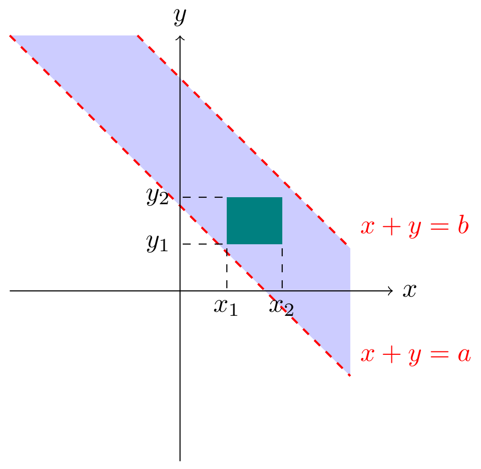
$$\mathrm{in}_{\mathrm{Add}^{-1}(I)}(x_1, y_1, x_2, y_2) \overset{def}{=} a < x_1 + y_1 < b \quad \wedge \quad a < x_2 + y_2 < b.$$

Since x_1, y_1, x_2, y_2 are all rational, this function is clearly recursive, and this, along with Theorem 4.1.22, gives us an effective open exhaustion for $\mathrm{Add}^{-1}(I)$.

□

Lemma 4.4.2. Let $f, g : \mathbb{R} \to \mathbb{R}$ be effectively open. Then, the function $(f \times g) : \mathbb{R} \times \mathbb{R} \to \mathbb{R} \times \mathbb{R}$ with $(f \times g)(x, y) = (f(x), g(y))$ is exhaustion-reflecting.

Figure 4.5: Visualization of a 2-cube contained in $\mathrm{Add}^{-1}((a,b))$



Proof. In order to prove this, we need to prove that for any open $U \subseteq \mathbb{R}\times\mathbb{R}$ with an open exhaustion, the set

$$(f\times g)^{-1}(U) = \{(x,y) \mid (f(x),g(y)) \in U\}$$

has an effective open exhaustion. It suffices to prove that for each 2-cube $I_1 \times I_2$,

$$(f\times g)^{-1}(I_1\times I_2) \ = \{(x,y) \mid (f(x),g(x)) \in I_1\times I_2\}$$

has an effective open exhaustion. We have

$$\begin{aligned}(f\times g)^{-1}(I_1\times I_2) &= \{(x,y) \mid (f(x),g(x)) \in I_1\times I_2\}\\ &= \{(x,y) \mid f(x)\in I_1 \wedge g(x)\in I_2\}\\ &= \{(x,y) \mid f(x)\in I_1\} \ \cap \ \{(x,y) \mid g(x)\in I_2\}\\ &= f^{-1}(I_1)\times\mathbb{R} \ \cap \ \mathbb{R}\times g^{-1}(I_2)\\ &= f^{-1}(I_1)\times g^{-1}(I_2)\end{aligned}$$

Since f, g are exhaustion-reflecting, and I_1, I_2 are open finite intervals and hence have open exhaustions, $f^{-1}(I_1)$ and $g^{-1}(I_2)$ both have open exhaustions respectively $(F_0, F_1, \ldots)$ and $(G_0, G_1, \ldots)$. Then $(F_0\times G_0, F_1\times G_1, \ldots)$ is an effective open exhaustion for $f^{-1}(I_1)\times g^{-1}(I_2)$.

□

Lemma 4.4.3. The function Diag : $\mathbb{R}\to\mathbb{R}\times\mathbb{R}$ with $\mathrm{Diag}(x) = (x,x)$ is exhaustion-reflecting.

Proof. In order to prove this, we need to prove that for any open $U \subseteq \mathbb{R}\times\mathbb{R}$ with an effective open exhaustion, the set

$$\mathrm{Diag}^{-1}(U) = \{x \mid (x,x)\in U\}$$

has an effective open exhaustion. We give an algorithm that outputs each stage of an effective open exhaustion for $\mathrm{Diag}^{-1}(U)$. For any $l\in\mathbb{N}$ of the given effective open exhaustion for U, the stage U_l consists of open 2-cubes

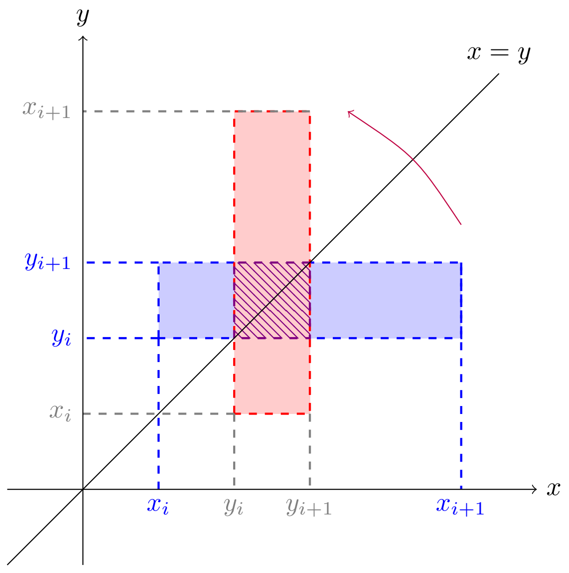
$$(x_1^l, x_2^l)\times(y_1^l, y_2^l), \ldots, (x_{2k_l-1}^l, x_{2k_l}^l)\times(y_{2k_l-1}^l, y_{2k_l}^l).$$

For any 2-cubes $(x_i^l, x_{i+1}^l) \times (y_i^l, y_{i+1}^l)$ we can construct an open interval

$$I_i^l \overset{def}{=} (x_i^l, x_{i+1}^l) \cap (y_i^l, y_{i+1}^l),$$

as shown[1] in Figure 4.6, and we can define $U_l \overset{def}{=} I_1^l, \ldots, I_{k_l}^l$. Then the sequence $(U_1, U_2, \ldots)$ is an open exhaustion for $\mathrm{Diag}^{-1}(U)$. □

Figure 4.6: Visualization of a 2-cube contained in $\mathrm{Diag}^{-1}(U)$



Theorem 4.4.4 (Addition). Let $f, g : \mathbb{R} \to \mathbb{R}$ be exhaustion-reflecting. Then $(f + g)$ is also exhaustion-reflecting.

Proof. Using Theorem 4.3.1, we know that compositions preserve exhaustion-reflectingness. Now we know that $(f+g)$ is composed from the functions Add, $(f \times g)$, and Diag. Therefore Lemmas 4.4.1, 4.4.2, and 4.4.3 complete the proof. □

4.4.3 Exhaustion-reflection for multiplication

Lemma 4.4.5. The function $\mathrm{Mult} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ with $\mathrm{Mult}(x, y) = x \cdot y$ is exhaustion-reflecting.

Proof. In order to prove this, we need to prove that for any open $U \subseteq \mathbb{R}$, the set

$$\mathrm{Mult}^{-1}(U) = \{(x, y) \mid x \cdot y \in U\}$$

has an effective open exhaustion. Now using Theorem 4.1.22 we only need to present a recursive function deciding if any arbitrary 2-cube $(x_1, x_2) \times (y_1, y_2)$ is in $\mathrm{Mult}^{-1}[I]$. Since $x \cdot y$ is a continuous function over the rectangle, its maximum value occurs at one of the corners. This means we can define

$$\begin{aligned} \mathrm{in}_{\mathrm{Mult}^{-1}(I)}(x_1, y_1, x_2, y_2) \overset{def}{=} & a < x_1 \cdot y_1 < b \ \wedge \ a < x_1 \cdot y_2 < b \ \wedge \\ & a < x_2 \cdot y_1 < b \ \wedge \ a < x_2 \cdot y_2 < b. \end{aligned}$$

[1] In this figure, the superscript l for endpoints of each interval is removed since we are only concerned with the lth stage at this point.

Since x_1, y_1, x_2, y_2 are all rational, this function is clearly recursive and this, along with Theorem 4.1.22, gives us an effective open exhaustion for $\text{Mult}^{-1}(I)$.

□

Theorem 4.4.6 (Multiplication). Let $f, g : \mathbb{R} \to \mathbb{R}$ be exhaustion-reflecting. Then $(f \cdot g)$ is also exhaustion-reflecting.

Proof. Using Theorem 4.3.1, we know that compositions preserve exhaustion-reflectingness. Now we know that $(f \cdot g)$ is composed from the functions $(f \times g)$, Diag, and Mult. Therefore Lemmas 4.4.2, 4.4.3, and 4.4.5 complete the proof. □

Theorem 4.4.7. Let $f : \mathbb{R} \to \mathbb{R}$ be an elementary function, then for any open set U with an effective open exhaustion, $f^{-1}(U)$ has an effective open exhaustion.

Proof. We define the effective open exhaustion for $f^{-1}(U)$ inductively using the following theorems:

- Base Cases:
  Theorem 4.2.1 (constant fuction), Theorem 4.2.2 (Identity) Theorem 4.2.3 (inverse),
  Theorem 4.2.4 (natural root),
  Theorem 4.2.5 (logarithm),
  Theorem 4.2.6 (exponential),
  Theorem 4.2.8 (sin),
  Theorem 4.2.9 (arcsin).
- Induction Steps:
  Theorem 4.3.1 (composition of functions),
  Theorem 4.4.4 (addition),
  Theorem 4.4.6 (multiplication).

□

Corollary 4.4.8. Let $f : \mathbb{R} \to \mathbb{R}$ be an elementary function, then for any open set U with an open exhaustion, $\mathbf{dom}(f) = f^{-1}(\mathbb{R})$ has an effective open exhaustion.

Chapter 5

Acceptability of Elementary Functions

In this section, we prove that all elementary functions are acceptable.

Recalling the Definition 2.5.4, a function $f : \mathbb{R} \to \mathbb{R}$ is *acceptable* if there exists a sequence X where:

(i) X is an effective open exhaustion for $\mathbf{dom}(f)$, and

(ii) f is effectively locally uniformly continuous w.r.t. X.

In the previous section, we proved that all elementary functions are exhaustion-reflecting, and hence have a domain with an effective open exhaustion. In this section, we complete the proof of acceptability of the elementary functions by proving that the elementary functions are effectively locally uniformly continuous with respect to the aforementioned exhaustion.

5.1 Preliminary Lemmas

The following theorem lets us use a simpler characterization for effective locally uniform continuity of a function that does not depend on any effective open exhaustion. Leveraging this characterization, we can reduce the problem of proving effective local uniform continuity w.r.t an open exhaustion for its domain, to a simpler problem. This characterization is especially used later in Theorems 5.4.1 and 5.5.1 for proving effective local uniform continuity of the addition and multiplication of functions.

Definition 5.1.1 (Local continuity witness). Let $f : \mathbb{R} \to \mathbb{R}$. A recursive function $N : \mathbb{Q} \times \mathbb{Q} \times \mathbb{N} \to \mathbb{N}$ is called a *local continuity witness for* f iff for any $a, b \in \mathbb{Q}$ with $[a, b] \subseteq \mathbf{dom}(f)$ and $k \in \mathbb{N}$, we have

$$\forall x, y \in (a, b) \quad |x - y| < 2^{-N(a,b,k)} \implies |f(x) - f(y)| < 2^{-k}.$$

Theorem 5.1.2 (Alternative characterization of effective local uniform continuity). Let $f : \mathbb{R} \to \mathbb{R}$ have an open domain with an open exhaustion $(U_1, U_2, \ldots)$. Then f is effectively locally uniformly continuous with respect to $(U_1, U_2, \ldots)$ if and only if there is a local continuity witness for f.

Proof. ($\Rightarrow$) Assuming f is effectively locally uniformly continuous with respect to an open exhaustion $(U_1, U_2, \ldots)$ for $\mathbf{dom}(f)$, we get a recursive function $M : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that for all $k, i \in \mathbb{N}$ and all $x, y \in U_i$,

$$|x - y| < 2^{-M(k,i)} \implies |f(x) - f(y)| < 2^{-k}.$$

We need to prove the existense of a local continuity witness for f. To do this, we present an algorithm computing a recursive function $N : \mathbb{Q} \times \mathbb{Q} \times \mathbb{N} \to \mathbb{N}$ that takes $a, b \in \mathbb{Q}$ and $k \in \mathbb{N}$ with $[a, b] \subseteq \mathbf{dom}(f)$ as inputs and returns a natural number such that

$$|x - y| < 2^{-N(a,b,k)} \implies |f(x) - f(y)| < 2^{-k}.$$

Since $(U_i)_{i\in\mathbb{N}}$ covers $\mathbf{dom}(f)$, there is a stage in which $[a, b]$ is covered. By definition of an effective open exhaustion, there is a recursive map which delivers the sequence of stages of $(U_i)_{i\in\mathbb{N}}$. Hence we can enumerate all stages until we find a stage s which contains $[a, b]$, and then we can output $N(a, b, k) = M(k, s)$. This guarantees that

$$\forall x, y \in (a, b) \quad |x - y| < 2^{-N(a,b,k)} \implies |f(x) - f(y)| < 2^{-k}.$$

($\Leftarrow$) Let us assume f has a local continuity witness $N : \mathbb{Q} \times \mathbb{Q} \times \mathbb{N} \to \mathbb{N}$. We need to define a recursive function $M : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that

$$\forall x, y \in U_i \quad |x - y| < 2^{-M(k,i)} \implies |f(x) - f(y)| < 2^{-k}.$$

Since we have an open exhaustion for $\mathbf{dom}(f)$, we have a recursive function listing the intervals in each stage. Let $(a_1, b_1, \ldots, a_{n_s}, b_{n_s})$ be the endpoints of the components of stage s. Let us enumerate the gaps between intervals as g_1, g_{n_s-1} with $g_j \in \mathbb{Q}$ for each $j \in \{1, \ldots, n_s - 1\}$. Then, we define

$$M(k, s) \overset{def}{=} \max(\{N(a_m, b_m, k) \mid 1 < m < n_s\} \cup \{\lceil 1/g_j \rceil \mid 1 \le j \le n_s - 1\}).$$

Consider $x, y \in U_s$. Then if $|x - y| < 2^{-M(k,s)}$, this means that x and y must be on the same interval in the open exhaustion. We also have that $|x - y| < 2^{-N(a_m,b_m,k)}$ for $1 < m < n_s$, this guarantees that $|f(x) - f(y)| < 2^{-k}$, and hence, proves that f is locally uniformly continuous with respect to $(U_i)_{i\in\mathbb{N}}$.

□

We take this theorem as justification for being able to talk about just "effective local uniform continuity", instead of having to talk about "effective local uniform continuity w.r.t an open exhaustion".

Theorem 5.1.2 gives us an alternative characterization of acceptable functions, i.e.:

Corollary 5.1.3. A function $f : \mathbb{R} \to \mathbb{R}$ is *acceptable* iff:

(i) The domain of f is the union of an effective open exhaustion, and

(ii) The function f has a local continuity witness.

Lemma 5.1.4. Let F : real × nat → real be a **WhileCC**-procedure approximating the function $f : \mathbb{R} \to \mathbb{R}$, with f monotone on its domain, then f is effectively locally uniformly continuous.

Proof. By Theorem 5.1.2, it suffices to give a recursive function $N : \mathbb{Q} \times \mathbb{Q} \times \mathbb{N} \to \mathbb{N}$ that for any $a, b, \in \mathbb{Q}$ with $[a, b] \subseteq \mathbf{dom}(f)$ satisfies

$$\forall x, y \in (a, b) \quad |x - y| < 2^{-N(a,b,k)} \implies |f(x) < f(y)| < 2^{-k}.$$

We present an algorithm to compute $N(a, b, k)$ for any $a, b \in \mathbb{Q}$ with $[a, b] \subseteq \mathbf{dom}(f)$. We begin by giving an informal description of the algorithm:

1. Start with a counter $c = 0$.
2. Divide $[a, b]$ into segments $[a_1, a_2], [a_2, a_3], \cdots, [a_{c_n-1}, a_{c_n}]$ of length at most 2^{-c}.
3. Check whether for all $i \in \{1, \ldots, c_n - 1\}$, there exists some $q_1 \in F^{\mathcal{R}}(a_i, c)$ and $q_2 \in F^{\mathcal{R}}(a_{i+1}, c)$ where
$$|q_2 - q_1| + 2^{-c-1} < 2^{-k}.$$
If this is not satisfied, increase c by one, and go to step 2. Otherwise, return c.

The algorithm we just defined computes the following functions:

$$\text{intervals}(a, b, i, c) = \begin{cases} [a + 2^{-c}i, a + 2^{-c}(i + 1)] & \text{if } 2^{-c}(i + 1) < b \\ [a + 2^{-c}i, b] & \text{otherwise} \end{cases}$$
$$\begin{aligned} N(a, b, k) = \min_{c\in\mathbb{N}} \ & \forall i \in \{0, \ldots, \lceil (b - a)/2^{-c} \rceil\} \\ & \exists q_1 \in F^{\mathcal{R}}(\text{intervals}(a, b, i, c), c) \\ & \exists q_2 \in F^{\mathcal{R}}(\text{intervals}(a, b, i + 1, c), c) \\ & |q_2 - q_1| + 2^{-c-1} < 2^{-k}. \end{aligned}$$

Intuitively, intervals(a, b, i, c) divides the interval $[a, b]$ into a finite set of sub-intervals of maximum length 2^{-c}, returning the ith such interval. Note that $|q_2 - q_1| + 2^{-c-1}$ is an overestimation of how much the value of f changes within the interval $[a_i, a_{i+1}]$. Since the function is monotone over I, the maximum change in any interval of length at least 2^{-c} is less than the estimation, and hence step 3 will guarantee that

$$\forall x, y \in (a, b) \quad |x - y| < 2^{-N(a,b,k)} \implies |f(x) - f(y)| < 2^{-k}.$$

□

5.2 Continuity for Basic Functions

Theorem 5.2.1 (Continuity for basic functions). The functions

- constant,
- identity,
- $\text{inv}(x) = 1/x$,
- log,
- $\exp(x) = e^x$,
- arcsin, and
- natural root

are effectively locally uniformly continuous w.r.t to their respective effective open exhaustions for their domains.

Proof. In each case, we can prove the alternative form of effective local uniform continuity defined in Theorem 5.1.2. All of the functions above are monotone on any closed interval in their domain, so we can use Lemma 5.1.4 to provide the requested recursive function for us in each case. □

Theorem 5.2.2 (Continuity for sin**).** The function $\sin(x)$ is effectively locally uniformly continuous w.r.t to the effective open exhaustion given in 4.2.8.

Proof. We know that the for any $x, y \in \mathbb{R}$, the inequality $|\sin(x) - \sin(y)| \leq |x - y|$ holds. Hence, we can simply define the function $M(k, l) = k$ for any $k, l \in \mathbb{N}$. Then for any $k, l \in \mathbb{N}$:

$$|x - y| < 2^{-k} \implies |\sin(x) - \sin(y)| \leq |x - y| < 2^{-k}$$

which gives us the effective local uniform continuity. □

5.3 Continuity for Composition of Functions

In this section, we prove that the composition of any two effectively locally uniformly continuous functions is effectively locally uniformly continuous w.r.t the domain of the composition. In order to prove this, we first need to prove that the following forward coverage property follows from uniform local continuity. Intuitively, we say a function satisfies the forwad coverage property if when applying the function on a point, based on which stage of the open exhaustion of the domain the point appears in, we can systematically anticipate the state of the open exhaustion for the range in which we can expect the value to fall in.

Definition 5.3.1 (Forward coverage property w.r.t some effective open exhaustion). Let $f : \mathbb{R} \to \mathbb{R}$, U an open set with an effective open exhaustion $X = (U_1, U_2, \ldots)$, and $f^{-1}(U)$ have an effective open exhaustion $(U'_1, U'_2, \ldots)$. Then f satisfies the *forward coverage property w.r.t* X, if there is a recursive function $S_f : \mathbb{N} \to \mathbb{N}$ where

$$x \in U'_l \implies f(x) \in U_{S_f(l)}.$$

Definition 5.3.2 (Interval forward coverage property w.r.t some open exhaustion). Let $f : \mathbb{R} \to \mathbb{R}$, U an open set with an effective open exhaustion $(U_1, U_2, \ldots)$. Then f satisfies the *interval forward coverage property*, if there is a recursive function $S_f : \mathbb{Q} \times \mathbb{Q} \times \mathbb{N} \to \mathbb{N}$ where

$$x \in (a, b) \implies f(x) \in U_{S_f(a,b,l)}$$

for any $a, b \in \mathbb{Q}$ and $[a, b] \subseteq f^{-1}(U)$.

Lemma 5.3.3. Let $f : \mathbb{R} \to \mathbb{R}$ be a function with interval forward coverage property with respect to some effective open exhaustion X. Then f has forward coverage property with respect to X as well.

Proof. Let U an open set with an effective open exhaustion $X = (U_1, U_2, \ldots)$, and $f^{-1}(U)$ have an effective open exhaustion $(U'_1, U'_2, \ldots)$. There is a recursive function $S_f : \mathbb{Q} \times \mathbb{Q} \times \mathbb{N} \to \mathbb{N}$ that, for any $a, b \in \mathbb{Q}$ and $[a, b] \subseteq U'_l$ and for any $x \in (a, b)$, satisfies

$$x \in (a, b) \implies f(x) \in U_{S_f(a,b,l)}.$$

Let the stage $U'_l = (a_1, b_1) \cup \cdots \cup (a_{n_l}, b_{n_l})$. Let us define

$$S(l) \overset{def}{=} \max\{S_f(a_1, b_1, l), \ldots S_f(a_{n_l}, b_{n_l}, l)\}$$

then clearly

$$x \in U'_l \implies f(x) \in U_{S(l)}.$$

This proves the forward coverage property. □

Definition 5.3.4 (Estimation of max and min over a closed interval). Let $f_0 : \mathbb{Q} \times \mathbb{N} \to \mathbb{Q}$ and $N : \mathbb{Q} \times \mathbb{Q} \times \mathbb{N} \to \mathbb{N}$. Then we define $\mathrm{Max}_{f_0,N} : \mathbb{Q} \times \mathbb{Q} \times \mathbb{N} \to \mathbb{Q}$ as

$$\mathrm{Max}_{f_0,N}(a, b, n) \overset{def}{=} \max\{f_0(q, n + 1) \mid q \in \{a_1, \ldots, a_m\}\},$$

and respectively, $\mathrm{Min}_{f_0,N} : \mathbb{Q} \times \mathbb{Q} \times \mathbb{N} \to \mathbb{Q}$ as

$$\mathrm{Min}_{f_0,N}(a, b, n) \overset{def}{=} \min\{f_0(q, n + 1) \mid q \in \{a_1, \ldots, a_m\}\},$$

where we divide $[a, b]$ into segments $[a_1, a_2], [a_2, a_3], \ldots, [a_{m-1}, a_m]$ such that

$$\forall i \in \{1, \ldots, m - 1\} \quad |a_{i+1} - a_i| < 2^{-N(a,b,n+1)-1}.$$

Theorem 5.3.5. Let $f : \mathbb{R} \to \mathbb{R}$ and $f_0 : \mathbb{Q} \times \mathbb{N} \to \mathbb{Q}$ satisfy, for any $n \in \mathbb{N}$ and any $x \in \mathbf{dom}(d) \cap \mathbb{Q}$,

$$|f_0(x, n) - f(x)| < 2^{-n}.$$

Let us also assume we have a recursive function $N : \mathbb{Q} \times \mathbb{Q} \times \mathbb{N} \to \mathbb{N}$ that for any $n \in \mathbb{N}$ and $a, b \in \mathbb{Q}$ with $[a, b] \subseteq \mathbf{dom}(f)$, we have

$$|x - y| < 2^{-N(a,b,n)} \implies |f(x) - f(y)| < 2^{-n}.$$

Then

$$\left|\mathrm{Max}_{f_0,N}(a, b, n) - \max_{x\in[a,b]} f(x)\right| < 2^{-n}$$

and

$$\left|\mathrm{Min}_{f_0,N}(a, b, n) - \min_{x\in[a,b]} f(x)\right| < 2^{-n}.$$

Proof. We prove

$$\left|\mathrm{Max}_{f_0,N}(a, b, n) - \max_{x\in[a,b]} f(x)\right| < 2^{-n}.$$

The case of minimum is similar.

Let us divide $[a, b]$ into segments $[a_1, a_2], [a_2, a_3], \ldots, [a_{m-1}, a_m]$, where $a_1 = a$ and $a_m = b$, such that

$$\forall i \in \{1, \ldots, m - 1\} \quad |a_{i+1} - a_i| < 2^{-N(a,b,n+1)-1}.$$

since $[a, b] \subseteq \mathbf{dom}(f)$ is a closed interval, we know that $f(x)$ will have its maximum in $[a, b]$. Since we have $[a, b] = [a_1, a_2], \ldots, [a_{m-1}, a_m]$, the maximum of $f(x)$ will occur at least in one interval $[a_j, a_{j+1}]$. Now:

$$\begin{aligned}
&\left|\text{Max}_{f_0,N}(a,b,n) - \max_{x\in[a,b]} f(x)\right| \\
&= \left|\max\{f_0(q,n+1) \mid q \in \{a_1,\ldots,a_m\}\} - \max_{x\in[a,b]} f(x)\right| \\
&= \left|\max\left\{ f_0(q,n+1) - \max_{x\in[a,b]} f(x) \mid q \in \{a_1,\ldots,a_m\}\right\}\right| \\
&= \left|f_0(a_j,n+1) - \max_{x\in[a_j,a_{j+1}]} f(x)\right| \\
&= \left|f_0(a_j,n+1) - f(a_j) + f(a_j) - \max_{x\in[a_j,a_{j+1}]} f(x)\right| \\
&\leq |f_0(a_j,n+1) - f(a_j)| + \left|f(a_j) - \max_{x\in[a_j,a_{j+1}]} f(x)\right| \\
&< 2^{-n-1} + 2^{-n-1} \\
&= 2^{-n}
\end{aligned}$$

□

Lemma 5.3.6. Let $f : \mathbb{R} \to \mathbb{R}$ be effectively locally multipolynomially approximable w.r.t. $(U_1, U_2, \ldots)$ by $(q_n)_{n\in\mathbb{N}}$ via M where $(U_1, U_2, \ldots)$ is an effective open exhaustion of $\mathbf{dom}(f)$. Then there is a function $f_0 : \mathbb{Q} \times \mathbb{N} \to \mathbb{N}$ such that

$$|f_0(x,n) - f(x)| < 2^{-n}.$$

Proof. Let us define $f_0 : \mathbb{Q} \times \mathbb{N} \to \mathbb{N}$ to be

$$f_0(x,n) \overset{def}{=} q_{M(n,\min\{l \mid x\in U_l\})}(x).$$

Then f_0 satisfies $|f_0(x,n) - f(x)| < 2^{-n}$ by the definition of effective local multipolynomial approximability. □

Theorem 5.3.7. All acceptable **WhileCC**-approximable functions satisfy the interval forward coverage property with respect to any effective open exhaustion.

Proof. Let $f : \mathbb{R} \to \mathbb{R}$ be an acceptable **WhileCC**-approximable function. Let U be an open set with an effective open exhaustion $(U_1, U_2, \ldots)$. Then, in order to prove that f satisfies the interval forward coverage property, we need to give a recursive function $S_f : \mathbb{Q} \times \mathbb{Q} \times \mathbb{N} \to \mathbb{N}$ that, for any $a, b \in \mathbb{Q}$ with $[a,b] \subseteq f^{-1}(U)$, satisfies

$$x \in (a,b) \implies f(x) \in U_{S_f(a,b,l)}.$$

By Fu and Zucker's equivalence Theorem 2.6.4 and Lemma 5.3.6, there is a function f_0 satisfying $|f_0(x,n) - f(x)| < 2^{-n}$. Since f is effectively locally uniformly continuous, we have a recursive function $N : \mathbb{Q} \times \mathbb{Q} \times \mathbb{N} \to \mathbb{N}$ that for any $n \in \mathbb{N}$ and $a, b \in \mathbb{Q}$ with $[a,b] \subseteq \mathbf{dom}(f)$, we have

$$|x - y| < 2^{-N(a,b,n)} \implies |f(x) - f(y)| < 2^{-n}.$$

Let us define

$$S_f(a,b,l) \overset{def}{=} \min\{n \mid [\mathrm{Min}_{f_0,N}(a,b,n), \mathrm{Max}_{f_0,N}(a,b,n)] \subseteq U_n\}.$$

Since $[a,b] \subseteq f^{-1}(U)$, it follows that $f([a,b]) \subseteq U$. Since f is continuous and defined on $[a,b]$, $f([a,b])$ must be a closed interval $[\min_{x\in[a,b]} f(x), \max_{x\in[a,b]} f(x)]$ which falls under some stage U_m. and since U_m is open, then by Theorem 5.3.5 there must be some $k \geq m$ such that

$$[\min_{x\in[a,b]} f(x), \max_{x\in[a,b]} f(x)] \subseteq [\mathrm{Min}_{f_0,N}(a,b,k) - 2^{-k}, \mathrm{Max}_{f_0,N}(a,b,k) + 2^{-k}] \subseteq U_m \subseteq U_k.$$

□

Corollary 5.3.8. All **WhileCC**-approximable, effectively locally uniform continuous functions satisfy the forward coverage property.

Proof. Follows directly from Theorem 5.3.7 and Lemma 5.3.3. □

Theorem 5.3.9 (Continuity for composition). Let $f, g : \mathbb{R} \to \mathbb{R}$ be **WhileCC**-approximable and effectively locally uniformly continuous w.r.t effective open exhaustions for their domains respectively. Let X be an effective open exhaustion for $\mathbf{dom}(f \circ g)$. Then the function $f \circ g$ is effectively locally uniformly continuous with respect to X.

Proof. Let $f, g : \mathbb{R} \to \mathbb{R}$ be **WhileCC**-approximable and effectively locally uniformly continuous with respect to effective open exhaustions $(U_n^g)_{n\in\mathbb{N}}$ and $(U_n^f)_{n\in\mathbb{N}}$ with recursive functions N_f, N_g respectively. Let $X = (U_1, U_2, \ldots)$ be an effective open exhaustion for $\mathbf{dom}(f \circ g)$. Then we need to show that there is a recursive function $N : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ that for $x, y \in U_l$ satisfies

$$|x - y| < 2^{-N(k,l)} \implies |f(g(x)) - f(g(y))| < 2^{-k}$$

making the function $f \circ g$ effectively locally uniformly continuous with respect to X. Let $x, y \in U_l$ for some $l \in \mathbb{N}$. Since $\mathbf{dom}(f \circ g) \subseteq \mathbf{dom}(g)$, for any $l \in \mathbb{N}$, the lth stage of our exhaustion for $\mathbf{dom}(f \circ g)$ is contained in some stage with index l' of the exhaustion for U^g. Note that the index l' can be found recursively by going through each stage of U^g and checking whether it contains all the intervals in U_l. Also, let l'' be the index of some stage of the exhaustion for U^f such that $g(U_l) \subseteq U^f_{l''}$. By Corollary 5.3.8, such l'' is recursively attainable from l. Let us define $N(k,l) \overset{def}{=} N_g(N_f(k,l''), l')$. Then

$$\begin{aligned} &|x - y| < 2^{-N(k,l)} = 2^{-N_g(N_f(k,l''),l')} \\ \implies &|g(x) - g(y)| < 2^{-N_f(k,l'')} \\ \implies &|f(g(x)) - f(g(y))| < 2^{-k} \end{aligned}$$

which gives us effective local uniform continuity of $f \circ g$. □

5.4 Continuity for Addition of Functions

Theorem 5.4.1 (Continuity for addition). Let $f, g : \mathbb{R} \to \mathbb{R}$ be effectively locally uniformly continuous with respect to effective open exhaustions for their domains, respectively. Then $(f + g)(x) = f(x) + g(x)$ is effectively locally uniformly continuous w.r.t. any effective open exhaustion for $\mathbf{dom}(f + g)$.

Proof. In this proof, we only work with the alternative characterization of effective local uniform continuity (Theorem 5.1.2). Let $f, g : \mathbb{R} \to \mathbb{R}$ be effectively locally uniformly continuous with recursive functions $N_f, N_g : \mathbb{Q} \times \mathbb{Q} \times \mathbb{N} \to \mathbb{N}$ in the alternative characterization, respectively. We need to present a recursive function $N : \mathbb{Q} \times \mathbb{Q} \times \mathbb{N} \to \mathbb{N}$ that for all $a, b \in \mathbb{Q}$ with $[a, b] \subseteq \mathbf{dom}(f + g)$ satisfies

$$|x - y| < 2^{-N(a,b,k)} \implies |(f + g)(x) - (f + g)(y)| < 2^{-k}$$

making the function $f + g$ effectively locally uniformly continuous.

Let us take an arbitrary $k \in \mathbb{N}$. For any $a, b \in \mathbb{Q}$, if $[a, b] \subseteq \mathbf{dom}(f + g)$ then $[a, b] \subseteq \mathbf{dom}(f)$ and $[a, b] \subseteq \mathbf{dom}(g)$. This means that the function

$$N(a, b, k) \overset{def}{=} N_f(a, b, k + 1) + N_g(a, b, k + 1) + 1$$

satisfies for $x, y \in (a, b)$

$$|x - y| < 2^{-N(a,b,k)} \implies |(f + g)(x) - (f + g)(y)| < 2^{-k}$$

since for any $x, y \in (a, b)$ with $|x - y| < 2^{-N(a,b,k)} = 2^{-(N_f(a,b,k+1)+N_g(a,b,k+1)+1)}$, we have

$$|x - y| < 2^{-N_f(a,b,k+1)},$$

and

$$|x - y| < 2^{-N_g(a,b,k+1)},$$

hence:

$$\begin{aligned}
|(f + g)(x) - (f + g)(y)| &= |f(x) + g(x) - f(y) - g(y)| \\
&\leq |f(x) - f(y)| + |g(x) - g(y)| \\
&< 2^{-k-1} + 2^{-k-1} \\
&= 2^{k}
\end{aligned}$$

□

5.5 Continuity for Multiplication of Functions

Theorem 5.5.1 (Continuity for multiplication). Let $f, g : \mathbb{R} \to \mathbb{R}$ be effectively locally uniformly continuous with respect to effective open exhaustions for their domains, respectively. Then $(f \cdot g)(x) = f(x) \cdot g(x)$ is effectively locally uniformly continuous w.r.t. any effective open exhaustion for $\mathbf{dom}(f \cdot g)$.

Proof. In this proof, we only work with the alternative characterization of effective local uniform continuity (Theorem 5.1.2). Let $f, g : \mathbb{R} \to \mathbb{R}$ be effectively locally uniformly continuous with recursive functions $N_f, N_g : \mathbb{Q} \times \mathbb{Q} \times \mathbb{N} \to \mathbb{N}$ in the alternative characterization, respectively. We need to present a recursive function $N : \mathbb{Q} \times \mathbb{Q} \times \mathbb{N} \to \mathbb{N}$ that for all $a, b \in \mathbb{Q}$ with $[a, b] \subseteq \mathbf{dom}(f \cdot g)$ satisfies

$$|x - y| < 2^{-N(a,b,k)} \implies |(f \cdot g)(x) - (f \cdot g)(y)| < 2^{-k}$$

making the function $f \cdot g$ effectively locally uniformly continuous.

Let us take an arbitrary $k \in \mathbb{N}$. For any $a, b \in \mathbb{Q}$, if $[a, b] \subseteq \mathbf{dom}(f \cdot g)$ then $[a, b] \subseteq \mathbf{dom}(f)$ and $[a, b] \subseteq \mathbf{dom}(g)$. We need to present some recursive function $N : \mathbb{Q} \times \mathbb{Q} \times \mathbb{N} \to \mathbb{N}$ satisfying for $x, y \in (a, b)$

$$|x - y| < 2^{-N(a,b,k)} \implies |(f \cdot g)(x) - (f \cdot g)(y)| < 2^{-k}.$$

We define the function N as follows:

$$N(a, b, k) \overset{def}{=} \max\{N_f(a, b, m), N_g(a, b, m)\}$$

where m is computed using

$$m = \min_{m_1 \in \mathbb{N}} \left\{ 2^{-m_1}(\text{Max}_{|f|}(a, b, 0) + \text{Max}_{|g|}(a, b, 0) + 2) < 2^{-k} \right\}$$

where $\text{Max}_{|f|}(a, b, 0)$ and $\text{Max}_{|g|}(a, b, 0)$ are overestimations for the maximum values of $|f|(x) = |f(x)|$ and $|g|(x) = |g(x)|$ on $[a, b]$ (Definition 5.3.4). Note that such an m must exist, and we can compute it by checking each natural number in ascending order. This gives us:

$$\begin{aligned}
|(f \cdot g)(x) - (f \cdot g)(y)| &= |g(x)f(x) - f(y)g(x) + f(y)g(x) - f(y)g(y)| \\
&\leq |g(x)(f(x) - f(y)) + f(y)(g(x) - g(y))| \\
&\leq |g(x)(f(x) - f(y))| + |f(y)(g(x) - g(y))| \\
&\leq 2^{-m}|g(x)| + 2^{-m}|f(y)| \\
&\leq 2^{-m}(|g(x)| + |f(y)|) \\
&\leq 2^{-m}(\text{Max}_{|g|}(a, b, 0) + \text{Max}_{|f|}(a, b, 0) + 2) \\
&< 2^{-k}. \qquad \text{(by defintion of } N(a, b, k))
\end{aligned}$$

□

Theorem 5.5.2. Let $f : \mathbb{R} \to \mathbb{R}$ be an elementary function, then f is effectively locally uniformly continuous w.r.t. any effective open exhaustion for its domain.

Proof. Follows directly from

- Theorem 5.2.1 (constant, identity, inverse, logarithm, exp, arcsine and natural root functions),
- Theorem 5.2.2 (sin function),
- Theorem 5.3.9 (composition of functions),
- Theorem 5.4.1 (addition of functions), and
- Theorem 5.5.1 (multiplication of functions).

□

Theorem 5.5.3. All elementary functions $f : \mathbb{R} \to \mathbb{R}$ are acceptable.

Proof. Follows directly from Theorem 4.4.7 and Theorem 5.5.2. □

Corollary 5.5.4. Using Theorem 3.12.1 and Theorem 5.5.3, along with the equivalence lemma 2.6.4, we can immediately conclude that any unary elementary function is also:

- GL-computable w.r.t. an effective open exhaustion for its domain,
- tracking computable, and
- multipolynomially approximable w.r.t. an effective open exhaustion for its domain.

Chapter 6

Conclusion and Future Work

The contributions of this thesis are as follows:

- We prove that all elementary functions are **WhileCC**-approximable.
- We prove that all elementary functions are acceptable.
- We present an alternative characterization of acceptable functions.

The following interesting problems are still left open:

- Determining the status of the generalized elementary functions to more than one argument: In particular, we would like to know if the generalized elementary functions are acceptable, considering the generalized version of acceptability given by Tucker and Zucker [2004]. Our idea of decomposing addition and multiplication in Section 4.4.1 seems to show us a way forward towards proving that non-unary elementary functions are acceptable.
- Determining the status of the While*-approximability model by Tucker and Zucker [1999], i.e., **WhileCC**-approximability in the absence of the countable choice operator: The nondeterministic choice seems to be an important feature in the **WhileCC** programming language. But we would also like to know how much power exactly we are adding to our language when adding this nondeterministic choice operator.
- Extending the equivalence theorem in Fu and Zucker [2014] to acceptable partial functions of type $\mathbb{R}^m \to \mathbb{R}$.

We also conjecture that all partial unary **WhileCC**-approximable functions are acceptable. We would also like to know the answer to the following questions:

- If the conjecture holds, are *non-unary* **WhileCC**-approximable functions acceptable?
- If the conjecture does not hold, what is a model of computation that characterizes exactly the class of acceptable functions?

Appendix A

Auxiliary Theorems

Theorem A.0.1 is used in both Sections 3.7 and 3.9, and Lemmas A.0.2, A.0.3, and A.0.4 are used in Section 3.7.

Theorem A.0.1 ([Spivak, 1994], page 308). For any arbitrary $\epsilon, a \in \mathbb{R}$ with $\epsilon > 0$, there exists some $n' \in \mathbb{N}$ such that, for all $n > n'$, we have $\frac{a^n}{n!} < \epsilon$.

Proof. Let $a, \epsilon \in \mathbb{R}$ with $\epsilon > 0$. Notice that for any arbitrary fixed $b \in \mathbb{R}$, and for any $m \in \mathbb{N}$, if $m \geq 2b$ then

$$\frac{b^{m+1}}{(m+1)!} = \frac{b}{m+1} \cdot \frac{b^m}{m!} < \frac{1}{2} \cdot \frac{b^m}{m!}.$$

Now, let us fix n_0 to be any natural number with $n_0 > 2a$. Then, we have:

$$\begin{aligned}
\frac{a^{n_0+1}}{(n_0+1)!} &< \frac{1}{2} \cdot \frac{a^{n_0}}{n_0!} \\
\frac{a^{n_0+2}}{(n_0+2)!} &< \frac{1}{2} \cdot \frac{a^{n_0+1}}{(n_0+1)!} < \frac{1}{2} \cdot \frac{1}{2} \cdot \frac{a^{n_0}}{n_0!} \\
&\vdots \\
\frac{a^{n_0+k}}{(n_0+k)!} &< \frac{1}{2^k} \cdot \frac{a^{n_0}}{n_0!}
\end{aligned}$$

Let 2^k be the first power of 2 for which $\frac{a^{n_0}}{(n_0)!\epsilon} < 2^k$ holds, then $\frac{a^{n_0+k'}}{(n_0+k')!} < \epsilon$ for any $k' > k$, which is the desired result. □

Lemma A.0.2. For any $x, y \in \mathbb{R}$, we have

$$e^x - e^{x-y} \leq e^{x+y} - e^x.$$

Proof.

$$\begin{aligned}
& e^x - e^{x-y} \leq e^{x+y} - e^x \\
\iff & e^x(1 - e^{-y}) \leq e^x(e^y - 1) \\
\iff & 1 - e^{-y} \leq e^y - 1 \\
\iff & 2 \leq e^y + e^{-y} \\
\iff & 1 \leq (e^y + e^{-y})/2 \\
\iff & 1 \leq \cosh(x) \\
\iff & \text{true}
\end{aligned}$$

□

Lemma A.0.3. Let the **WhileCC**-procedure $F : \mathsf{real} \times \mathsf{nat} \to \mathsf{real}$ approximate $f : \mathbb{R} \to \mathbb{R}$, $x \in \mathbf{dom}(f)$, and $r \in F^{\mathcal{R}}(x, m)$. Then $\left|e^{f(x)} - e^r\right| < \left|e^f(x) - e^{f(x)+2^{-m}}\right|$.

Proof. We prove this by cases:

- $r \leq f(x)$:

$$\begin{aligned}\left|e^{f(x)} - e^r\right| &< \left|e^{f(x)} - e^{f(x)-2^{-m}}\right| && \text{(since } e^r > e^{f(x)-2^{-m}}\text{)}\\ &\leq \left|e^{f(x)} - e^{f(x)+2^{-m}}\right| && \text{(by Lemma A.0.2)}\end{aligned}$$

- $r > f(x)$:

$$\begin{aligned}\left|e^{f(x)} - e^r\right| &= \left|e^r - e^{f(x)}\right|\\ &< \left|e^{f(x)+2^{-m}} - e^{f(x)}\right| && \text{(since } e^r < e^{f(x)+2^{-m}}\text{)}\\ &= \left|e^{f(x)} - e^{f(x)+2^{-m}}\right|\end{aligned}$$

□

Lemma A.0.4. Let $x \in \mathbb{R}$ and $N \in \mathbb{N}$. Then for any $k \in \mathbb{N}$

$$|2x| < N \implies \frac{|x|^{N+k}}{(N+k)!} < \frac{1}{2^k}\frac{|x|^N}{N!}.$$

Proof. The proof is inspired by the proof of Theorem A.0.1.

$$\begin{aligned}|2x| < N &\implies \frac{|x|}{N} < \frac{1}{2}\\ &\implies \forall k \in \mathbb{N} \quad \frac{|x|^k}{(N+1)\cdots(N+k)} < \frac{1}{2^k}\\ &\implies \forall k \in \mathbb{N} \quad \frac{|x|^N}{N!} \cdot \frac{|x|^k}{(N+1)\cdots(N+k)} < \frac{1}{2^k} \cdot \frac{|x|^N}{N!}\\ &\implies \forall k \in \mathbb{N} \quad \frac{|x|^{N+k}}{(n+k)!} < \frac{1}{2^k} \cdot \frac{|x|^N}{N!}\end{aligned}$$

□

Bibliography

A. Bauer. Notes on realizability, 2022. URL https://www.andrej.com/zapiski/MGS-2022/notes-on-realizability.pdf. Lecture notes from Mathematical Graduate Seminar, 2022 (Last accessed: 2025-03-20). 16

C. Böhm and G. Jacopini. Flow diagrams, turing machines and languages with only two formation rules. *Commun. ACM*, 9(5):366–371, May 1966. ISSN 0001-0782. doi:10.1145/355592.365646. 14

N. Cutland. *Computability, an Introduction to Recursive Function Theory*. Cambridge University Press, New York, 1980. 14

M. Q. Fu and J. Zucker. Models of computation for partial functions on the reals. *Journal of Logical and Algebraic Methods in Programming*, 84(2):218–237, 11 2014. ISSN 2352-2208. doi:10.1016/j.jlamp.2014.11.001. iii, 1, 13, 14, 17, 18, 19, 20, 21, 76

M. B. Pour-El and J. I. Richards. *Computability in analysis and physics*. Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1989. ISBN 3-540-50035-9. 19, 21

M. Spivak. *Calculus*. Cambridge University Press, third edition, 1994. 77

V. Stoltenberg-Hansen and J. V. Tucker. Concrete models of computation for topological algebras. *Theoretical Computer Science*, 219(1-2):347–378, 1999. ISSN 0304-3975. doi:10.1016/S0304-3975(98)00296-5. 1

M. Tenenbaum and H. Pollard. *Ordinary Differential Equations: An Elementary Textbook for Students of Mathematics, Engineering, and the Sciences*. Dover Books on Mathematics. Dover Publications, 1985. ISBN 9780486649405. URL https://books.google.ca/books?id=iU4zDAAAQBAJ. 22, 23

J. Tucker and J. Zucker. Computable total functions on metric algebras, universal algebraic specifications and dynamical systems. *The Journal of Logic and Algebraic Programming*, 62(1):71–108, 2005. ISSN 1567-8326. doi:10.1016/j.jlap.2003.10.001. 1, 18, 19

J. V. Tucker and J. I. Zucker. Computation by 'While' programs on topological partial algebras. *Theoretical Computer Science*, 219(1):379–420, 1999. ISSN 0304-3975. doi:10.1016/S0304-3975(98)00297-7. iii, 1, 76

J. V. Tucker and J. I. Zucker. Computable functions and semicomputable sets on many-sorted algebras. *Handbook of logic in computer science*, 5:317–523, 2001. 3, 4, 5, 11

J. V. Tucker and J. I. Zucker. Abstract versus concrete computation on metric partial algebras. *ACM Trans. Comput. Log.*, 5(4):611–668, 2004. doi:10.1145/1024922.1024924. iii, 1, 3, 5, 9, 10, 11, 12, 13, 18, 76

Index