박 사 학 위 논 문
Ph.D. Dissertation

검증된 계산을 위한 연속적 추상형 자료형

Continuous Abstract Data Types
for Verified Computation

2021

박 세 원 (朴 世 原 Park, Sewon)

한 국 과 학 기 술 원

Korea Advanced Institute of Science and Technology

박 사 학 위 논 문

검증된 계산을 위한 연속적 추상형 자료형

2021

박 세 원

한 국 과 학 기 술 원

전산학부

검증된 계산을 위한 연속적 추상형 자료형

박 세 원

위 논문은 한국과학기술원 박사학위논문으로
학위논문 심사위원회의 심사를 통과하였음

2021년 6월 7일

심사위원장	Martin Ziegler	(인)
심 사 위 원	강 지 훈	(인)
심 사 위 원	양 홍 석	(인)
심 사 위 원	최 성 희	(인)
심 사 위 원	Alex Simpson	(인)
심 사 위 원	이 계 식	(인)

Continuous Abstract Data Types for Verified Computation

Sewon Park

Advisor: Martin Ziegler

A dissertation submitted to the faculty of
Korea Advanced Institute of Science and Technology in
partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science

Daejeon, Korea
June 7, 2021

Approved by

Martin Ziegler
Professor of School of Computing

The study was conducted in accordance with Code of Research Ethics[1].

[1] Declaration of Ethical Conduct in Research: I, as a graduate student of Korea Advanced Institute of Science and Technology, hereby declare that I have not committed any act that may damage the credibility of my research. This includes, but is not limited to, falsification, thesis written by someone else, distortion of research findings, and plagiarism. I confirm that my thesis contains honest conclusions based on my own careful research under the guidance of my advisor.

DCS 박세원. 검증된 계산을 위한 연속적 추상형 자료형. 전산학부 . 2021년. 148+v 쪽. 지도교수: 지글러마틴. (영문 논문)
Sewon Park. Continuous Abstract Data Types for Verified Computation. School of Computing . 2021. 148+v pages. Advisor: Martin Ziegler. (Text in English)

초 록

이 논문에서는 실수를 추상형 자료형으로 제공하는 명령형 프로그래밍 언어들을 고안한다. 실수라는 연속적인 자료를 추상적으로 다루는 기존의 방식으로는 크게 대수적 계산 모형에 기반하는 것과 실수를 double 등의 유한 표기법을 사용하여 근사하는 것이 있다. 대수적 계산 모형의 경우 실제로 컴퓨터에 구현할 수 없거나 초월수 등을 다룰 수 없다는 한계가 있으며 유한 표기법에 기반한 계산의 경우 엄밀한 계산 모형이 없거나 계산 결과를 신뢰할 수 없다는 문제가 존재한다. 이 논문에서는 연속형 자료에 대한 계산의 엄밀한 이론적 토대인 계산 해석학에 기반하여 위의 문제들을 해결한다. 이 논문에서 제안되는 언어들은 명령형 프로그래밍 언어로 선행 조건과 후행 조건을 통해 프로그램들을 명세할 수 있다. 또한, 이 논문에서는 명세된 프로그램들을 검증할 수 있는 호아르 방식의 프로그래밍 검증법을 정의한다. 결과적으로 사용자들은 쉽게 실수 계산을 프로그램으로 표현할 수 있으며, 작성된 프로그램의 작동 방식을 명세할 수 있고, 명세된 프로그램을 검증할 수 있다. 프로그래밍 언어들을 고안하는 단계를 프레임워크로 만들어서 행렬, 연속함수 등 다른 연속적 추상형 자료형으로 확장하는 방법 또한 제안한다.

핵 심 낱 말 실수 계산, 연속적 추상형 자료형, 계산 해석학, 명령형 언어, 프로그램 검증

Abstract

We devise imperative programming languages for verified real number computation where real numbers are provided as abstract data types such that the users of the languages can express real number computation by considering real numbers as abstract mathematical entities. Unlike other common approaches toward real number computation, based on an algebraic model that lacks implementability or transcendental computation, or finite-precision approximation such as using double precision computation that lacks a formal foundation, our languages are devised based on computable analysis, a foundation of rigorous computation over continuous data. Consequently, the users of the language can easily program real number computation and reason on the behaviours of their programs, relying on their mathematical knowledge of real numbers without worrying about artificial roundoff errors. As the languages are imperative, we adopt precondition-postcondition-style program specification and Hoare-style program verification methodologies. Consequently, the users of the language can easily program a computation over real numbers, specify the expected behaviour of the program, including termination, and prove or disprove the correctness of the specification. Furthermore, we suggest extending the languages with other interesting continuous data, such as matrices, continuous real functions, et cetera.

Keywords real number computation, continuous abstract data type, computable analysis, imperative programming, formal verification

Contents

List of Tables

List of Figures

Chapter 1. Introduction

Real numbers are infinite objects that cannot be represented exactly using discrete data. Traditional models of computation are based on discrete data such as natural numbers, integers, rational numbers, discrete graphs, and et cetera. Turing machines, which is the most renowned model, read and write finite sequences of finite alphabets. When we abstract Turing machines, they represent only partial functions from natural numbers to natural numbers. Hence, any attempt on computing over real numbers using Turing machines, or any equivalent models, fail.

`double`-like data types in common programming languages, is a superstition. Its success in practical applications is misleading too many people. It disguises itself as it realizes real number computation. The set of finite-precision floating-point numbers is countable, and obviously, it fails to realize real number computation. It is not hard to find where a finite-precision floating-point computation becomes erroneous. When we look at any programming communities, there always are questions like "why 1.0+1.0 is not equal to 2?" And, there always are answers like "that is how real number computation is". Due to its design, rounding errors are inherent in floating-point computation [Gol91, Rum88, LW02, KMP+08]. When it comes to deciding the order of two real number expressions, it is possible to compute the totally wrong answer due to the errors in evaluating the two operands. In mission-critical applications, the errors will not be a Q&A post that we can laugh at [Hol94, JM97].

Algebraic models such as Blum-Shub-Smale machines [BSS+89] or Real-RAM [Sha78] consider algebraic real numbers. Algebraic real numbers are those that are roots of integer polynomials. Being represented by integer polynomials, even discrete models of computation such as Turing machines can compute over them exactly. However, in many scientific computing applications, the domain of computation often exceeds from being algebraic: π, e, and et cetera. Extending the domain, the assumption on comparing real numbers get unrealistic [Bra03, BV99]. Hence, though they surely are one important aspect of computing over real numbers, they are not in the scope of this topic. Hence, when we mention real number computation in this dissertation, it refers to computing over not only algebraic but all real numbers.

When there is a set that we want to compute over, say A, the set itself is not what machines recognize. When we abstract Turing machines as partial functions from $\mathbb{N}$ to $\mathbb{N}$, to compute over A using Turing machines, there should be some representation $r : \mathbb{N} \rightharpoonup A$. A representation is a relation of realization. If $r(n) = a \in A$ holds for some $n \in \mathbb{N}$, the natural number n can be seen as an implementation of the abstract object $a \in A$. Then, we can let a Turing machine $\mathcal{M} : \mathbb{N} \rightharpoonup \mathbb{N}$ realizes or computes a mathematical function $f : A \to A$ with regards to the representation r if for every element $a \in A$ and its implementation n, it holds that $r\big(\mathcal{M}(n)\big) = f(a)$. A representation is valid if it, as a function, is surjective. I.e., a valid representation must make every element implementable. In this sense, as the set of real numbers' cardinality strictly exceeds the set of natural numbers' cardinality, it is impossible to work with ordinary Turing machines to compute over the set of real numbers. There is no valid representation on the set of real numbers for the ordinary Turing machines.

Hence, to 'rigorously' compute over the set of real numbers or any other sets whose cardinality exceeds the cardinality of the set of natural numbers, it is necessary to extend the model of computation and the notion of representation. Type-2 Turing machines, though we will not go in deep into its definition yet, provides a formal framework for computing over the set of infinite sequences $\mathbb{N}^{\mathbb{N}}$. Intuitively, they

are extensions of Turing machines by letting the input and output tapes be infinitely long. Abstracting it, type-2 machines represent partial functions from $\mathbb{N}^{\mathbb{N}}$ to $\mathbb{N}^{\mathbb{N}}$. Having the domain of data extended to $\mathbb{N}^{\mathbb{N}}$, (infinite) representation on a set A is, now, a surjective function $\delta : \mathbb{N}^{\mathbb{N}} \rightharpoonup A$. And, an infinite sequence $\varphi \in \mathbb{N}^{\mathbb{N}}$ is an implementation of $a \in A$ with regards to the representation if $\delta(\varphi) = a$ holds. Similarly, the notion of computing a function on type-2 machines is extended.

The right way of computing over real numbers is, as opposed to using finite-precision floating-point computation, to use infinite representation of real numbers such that it avoids any rounding errors [BCRO86, BC88, PER89, DG93, EE00, Wei00, Mül00, EHS04, BCC$^+$06, CNR11, KTD$^+$13]. In fact, there are many, infinitely many, representations of real numbers; since any surjective function $\delta : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{R}$ is a representation. Among the infinitely many others, let us see what the *Cauchy representation*, which is often called the *standard representation*, is. The set of rational numbers $\mathbb{Q}$ being countable, we can embed it in the set of natural numbers $\iota_{\mathbb{Q}} : \mathbb{N} \to \mathbb{Q}$ such that all primitive operations of rational numbers are computable. A datum $\varphi \in \mathbb{N}^{\mathbb{N}}$ implements a real number x if n'th entry of the datum $\varphi(n)$ represents a 2^{-n} approximation to the real number x. In other words, δ_{Cauchy} is

$$\delta_{\text{Cauchy}}(\varphi) = x :\Leftrightarrow |\iota_{\mathbb{Q}}(\varphi(n)) - x| \leq 2^{-n} \quad \text{for all natural number } n \,.$$

A datum φ is an implementation of $x \in \mathbb{R}$ if φ encodes a sequence of rational numbers which converges rapidly to x. The datum φ can be seen as a magical black box that when we inquire some portion $n \in \mathbb{N}$ of the real number that it represents, it tells us the amount of information. The crucial part of it is that we can repeat the inquiry as many times as we want and ask as much information out of it. Under the Cauchy representation, the field arithmetic of real numbers can be computed *exactly*.

On the first day, there were Turing machines. When we make Java, C, C++, or Python programs, it is not too exaggerating to say that Turing machines are behind them. Turing machines provide a formal foundation for computing over any discrete objects such as natural numbers, integers, and so on. Abstracting Turing machines away, we get programming languages. Only their tedious semantics concern how to translate each instruction of the programming language to Turing machines' instructions. Lucky us, and all the programmers, we do not have to worry about Turing machine instructions when we write a simple Python program.

The same is desired for type-2 computation. Type-2 machines provide a formal foundation for computing over any sets whose cardinality is Continuum. To make it practically usable, it is essential to abstract the implementation-specific details away. We need to have a programming language that works at the abstract level. How programs in the language to be actually simulated by machines should be hidden to the users; thus, it is okay for them to be aware of only the abstract semantics. And, that is what this dissertation is all about, especially for real numbers.

We can introduce real numbers in a programming language as a primitive notion of the language, such as int, double, et cetera [DG93, Esc96, ES14]. This approach, which is classified to be *external* by [BES02a], is suitable when we are interested in the abstraction of real numbers. However, declaring, not defining, a type for real numbers, we have to face a vital decision problem: which real number operations should come as primitives and what should their semantics be. To make the decision, we need to know the universal property of real number computation as we do not want our data type for real numbers to be representation specific.

Using the Cauchy representation, the field arithmetic of real numbers is computed exactly. Given an arithmetical expression e, when we compute it, the number that the computer returns is exactly the value that e mathematically represents. (Of course, the real number itself does not get printed; when we

put an additional input n, we get 2^{-n} approximation to the real number.) However, it cannot be decided if two real numbers are identical in this approach. Suppose we have two sequences of rational numbers $(q_n)_{n\in\mathbb{N}}$ and $(r_n)_{n\in\mathbb{N}}$ that converge to real numbers x and y, respectively. The real numbers are identical if and only if $\forall n.\ |q_n - r_n| \leq 2^{-n}$ holds. And, testing if the statement holds is not decidable. The best we can get is to say "no" when there is an index that n such that $|q_n - r_n| > 2^{-n}$ holds, by testing it for each n. In consequence, testing inequality of real numbers $x < y$ is only partially computable in the sense that when $x \neq y$, we can say if $x < y$ or $y < x$. However, when $x = y$, the procedure of testing $x < y$ will run into an infinite loop and never terminates. The explanation thus far is based on the Cauchy representation. However, this is a universal property of infinite representation in that whichever representation we use, inequality tests can only be computed partially [Wei00, Theorem 4.1.16].

As inequality tests become partial, nondeterminism becomes essential in real number computation . [Luc77] [1] Suppose, in the middle of some computation, we have to decide if a real number x is positive or not. The whole computation fails when x happens to be precisely 0 at the moment. However, in many cases, it is not crucial to decide the sign of x precisely. (The sub-procedure being imprecise does not necessarily mean that the whole computation being inexact.) Instead, if we are given a tolerance factor $\epsilon > 0$, we compute the sign of x within the error: when $|x| \geq \epsilon$, compute the sign of x correctly, and if $-\epsilon < x < \epsilon$, return *nondeterministically* either $+1$ or -1. The tolerance factor, being an additional input, can be seen as the required preciseness of the approximation. The smaller it gets, the more accurate the total program gets. The nondeterminism can realized by the parallel evaluations: evaluate $x > -\epsilon$ and $x < \epsilon$ in parallel, return $+1$ if testing $x > -\epsilon$ succeeds, and return -1 if testing $x < \epsilon$ succeeds. Since x cannot be $-\epsilon$ and $+\epsilon$ at the same time, at least one of the two tests successes. It is nondeterministic since when both hold, we do not know in prior which does terminate first.

There are different approaches to equip a programming language for real number computation with nondeterminism. The first is to let the language itself be nondeterministic. It is possible to create arbitrary nondeterministic branches: either binary, finite, or countable. Following this approach, we can benefit from well-studied nondetermistic languages such as [Dij75, Apt83, AP86, Nel89]. This approach can be found in [Mül00, KTD+13]. The other approach is to restrict the use of nondeterminism only for real number comparisons. For example, in [BH98, MRE07], the language itself cannot create an arbitrary nondeterministic branch. Nondeterminism happens only for the comparison of real numbers with a tolerance factor as in the above paragraph. In this dissertation, we follow the first approach. Thus, our languages become more expressive. And, we use the well-studied theory of nondeterministic programming languages.

In order to go beyond algebraic computation, we need to equip our languages with the functionality of computing real numbers by the limits of sequences that approximate them. The set of infinite sequences of real numbers $\mathbb{R}^{\mathbb{N}}$ admits representations. Deferring discussions on correct representations of $\mathbb{R}^{\mathbb{N}}$ to later, let us say $\delta^{\omega}_{\text{Cauchy}}$ be the representation which is defined by the Cantor-style encoding:

$$\delta^{\omega}_{\text{Cauchy}}(\varphi) = (x_i)_{i\in\mathbb{N}} :\Leftrightarrow \delta_{\text{Cauchy}}(\varphi_i) = x_i \text{ for all } i \in \mathbb{N} \text{ where } \varphi_i := \big(\varphi((i+n)\cdot(i+n+1)/2+i)\big)_{n\in\mathbb{N}}\,.$$

With the representation of infinite sequences of real numbers, of course, projections are computable.

[1] It is also called *nonextensional computation* [Bra95] to make it distinguished from computation done by nondeterministic machines in complexity theory [Zie05]. Anyhow, we stick to the terminology nondeterminism in this dissertation that is from programming language theory [Dij75, AP86]

More importantly, the partial operation

$$\begin{array}{rcccc} \mathsf{lim} & : & \{(x_i)_{i\in\mathbb{N}} \mid \exists z.\, \forall n.\, |z - x_n| \leq 2^{-n}\} & \to & \mathbb{R} \\ & : & (x_i)_{i\in\mathbb{N}} & \mapsto & \lim_{n\to\infty}(x_i) \end{array}$$

computing the limits of rapid Cauchy sequences is computable. (We say a Cauchy sequence rapid if it has the above rate of convergence, similarly as in [Bis67].) We have not yet defined what it means to compute a partial function. And, we are not going into detail about it in this introduction section. However, the message is this: using the representation approach, when we can compute a rapid Cauchy sequence, we can construct the real number that is the limit of the sequence. For example, if we have a procedure $f : \mathbb{N} \to \mathbb{R}$ where $f(n)$ is a 2^{-n} approximation to π, we can transform the procedure f to be the real number π.

Any programming language that provides a primitive type for real numbers should have the properties: exact field arithmetic, partial comparison test, nondeterministic branching, and construction of real numbers by the limits of rapid Cauchy sequences. These requirements have to be reflected through the program's constructs and formal semantics. Also, at the same time, it is necessary that the language is not representation specific such that the users of the language can write their programs and reason on their programs' behaviours relying on their mathematical knowledge of real numbers.

One crucial reason to follow the representation approach for real number computation is reliability. Imperative programming, not only for its wide usage in practical scientific computing, has its advantage in verification. Its well-studied precondition-postcondition-style program specification and Hoare-style program verification methodologies make it easier for the users to specify the behaviour of their programs and prove the correctness of the programs [Apt81, Kle99, vO01, AO19]. Hence, in order to benefit from the well-founded theory and from the potential applicability, it is desired to devise an imperative language and its verification principles for real number computation.

Wrapping up the motivation thus far, in this dissertation, we devise imperative languages that provide data types and operations for real number computation. They are equipped with formal semantics that well-reflects the characteristics of real number computation explained thus far, and verification calculi for reasoning on the behaviours of real number computations.

The Overview and Organization of this Dissertation We want a language with a firm theoretical background. In Chapter 2, the category of assemblies over Kleene's second algebra is introduced, which is a universe of continuous data and computable functions between them. The purpose of the chapter is first to introduce the preliminary concepts of the dissertation, computable analysis, which is a foundation for continuous computation, and to define the four important endofunctors:

1. $\flat$ a functor for expressing partial functions that diverge out of their domains,
2. $\sharp$ a functor for expressing partial functions that eventually abort out of their domains,
3. $\natural$ a functor for expressing any partial functions whose behaviours out of their domains are not specified, and
4. M a functor for expressing (nondeterminitsic) multifunctions.

The functors are used throughout this dissertation. Using the functors, we express various types of computable partial multifunctions which are the atomic notions in our programming languages. In

Chapter 3, we devise the programming language called ERC by extending a simple imperative programming language with the data type R for real numbers and their operations. Its denotational semantics is defined using Plotkin powerdomain, which is for finite nondeterminism. We conclude the chapter by introducing a sound verification calculus. In Chapter 4, we define an interpreter of ERC to the category of assemblies over Kleene's algebra. In the interpretation, the functors are intensively used. By having the interpretation, we show that the language is implementable and the semantics is computable. However, to make the language as simple as possible, ERC does not provide a functionality to compute the limits of (rapidly) converging sequences inside its programs. Chapter 5, we extend the language with an explicit limit operator and define a new language called Clerical. We define its denotational semantics and a sound and relatively complete verification calculus. Chapter 6, by interpreting Clerical in the category of assemblies, shows that the language's semantics is computable.

Let us go trough the organization of each chapter.

Chapter 2 In this chapter, we recap the theory of computation over continuous data. In Section 2.1, we recap type-1 computation, that is, computation over discrete data that can be represented by $\mathbb{N}$ and be processed by ordinary Turing machines. In Section 2.2, type-2 computation over $\mathbb{N}^{\mathbb{N}}$ is introduced mostly based on [Wei00]. In the section, we classify computable partial functions from $\mathbb{N}^{\mathbb{N}}$ to $\mathbb{N}^{\mathbb{N}}$ which is a subclass of continuous partial functions with regards to the standard topology on $\mathbb{N}^{\mathbb{N}}$. Section 2.3 introduces the category of represented set Rep. A represented set is a pair (A, δ_A) of a set of Continuum cardinality A and a partial surjective function $\delta : \mathbb{N}^{\mathbb{N}} \rightharpoonup A$ called representation. Computable functions are defined between represented sets, and the collection of represented sets and computable functions is Rep. In the section, we go through various represented sets, including $\mathbf{R}_{\mathrm{Cauchy}}$ the standard represented set of real numbers. In Section 2.3.2, we define two very important functors $\flat, \sharp : \mathsf{Rep} \to \mathsf{Rep}$ which are used to express certain classes of computable partial functions in Rep. However, we conclude that the category Rep is not suitable to deal with general partial functions. In Section 2.4, the definitions of applicative functors, monads, and the corresponding function liftings are briefly introduced. Then, we analyze our functors $\flat$ and $\sharp$ accordingly.

In Section 2.5, we study the computational structure of $\mathbb{R}$. In Section 2.5.1, we define a notion called *nc* and show some represented sets of real numbers are *nc*, which we do not want to have. In Section 2.5.2, we show that any represented set of real numbers that makes the ordinary lim computable is *nc*. Further recapping noncomputability results from computable analysis, we justify the choice of the standard representation of real numbers.

In Section 2.6, the concept of multifunction, which is the nondeterminism we deal with in computable analysis, is introduced. In the section, the notion of computing multifunctions is introduced. And, some important examples, including computing the soft signs of real numbers, are demonstrated. Again, we conclude the section that Rep is not suitable to deal with multifunctions.

With the above motivations that Rep is not suitable for general partial functions and multifunctions, in Section 2.7, the category of assemblies over Kleene's second algebra (the category of assemblies in short) is introduced. The category $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ is a generalization of Rep where general computable partial functions and computable multifunctions appear as morhpisms. In Section 2.7.1, we extend the previous functors to $\flat, \sharp : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ and define a functor $\natural : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ which is for general computable partial functions. In Section 2.7.2, we define a functor $\mathsf{M} : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ which is for computable multifunctions. Hence, we can refer to a computable partial multifunction simply by a morhpism $f : \mathbf{A} \to \natural\mathsf{M}(\mathbf{B})$ in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$. In Section 2.7.3, we show that $\natural, \sharp, \mathsf{M}$ are countably applicative

in that for a morphism on sequence $f : \mathbf{A}^\omega \to \mathbf{B}$, we can naturally extend it to $f^\dagger : (\mathsf{M}(\mathbf{A}))^\omega \to \mathbf{B}$. Using this property, at the end of the section, we construct a morphism that computes the absolute value function.

Chapter 3 In this chapter, we define our first imperative language ERC that stands for Exact Real Computation. At the beginning of the chapter, we introduce its design choices. Instead of providing Booleans as its primitive data type, ERC provides its lazy lifting whose denotation is $\mathbb{K} = \{tt, ff, uk\}$ as its primitive type. The operations on $\mathbb{K}$ are that of Kleene's three value logic. Hence, we call $\mathbb{K}$ Kleenean where uk is the third truth value.

In Section 3.2, we define the formal syntax and the type system of ERC. In Section 3.3, we define the denotational semantics of ERC. To each program in ERC, we define its denotation as a set-valued set-theoretic function. As it is essential in real number computation, ERC provides finite nondetermistic branchings. Therefore, we take Plotkin powerdomain $\mathbb{P}(\square_\perp)$ [Plo76] to define our denotational semantics. By using a fixed-point theorem, we prove the well-definiteness of our denotational semantics, including that of while loops. At the end of the section, we prove that ERC is complete in the sense that any computable partial real function is expressible in ERC.

In Section 3.4, we define verification principles of ERC. In Section 3.4.1, we define a logical language, which we call the logic of ERC, that is used to specify the behaviour of ERC programs. In the section, we prove that the language is expressive enough to define the denotational semantics of the term language of ERC. For any given programming term t, there is a formula in the logical language that defines the denotation of t. We prove that the logical language is complete and decidable by being a safe mixture of Presburger arithmetic and real closed field. In Section 3.4.2, we define precondition-postcondition-style program specification for ERC, and devise Hoare-style proof rules for correct specifications. At the end of the section, we prove that the devised proof rules are sound with regards to our denotational semantics leaving a remark that the verification calculus cannot be complete.

Chapter 4 In this chapter, we prove the computability and implementability of ERC and suggest extending ERC with other continuous data. In Section 4.1, we define an abstract interpreter for ERC. The interpreter maps each ERC program to a morphism in $\mathsf{Asm}(\mathbb{N}^\mathbb{N})$. In Section 4.1.1, we construct a monad $\mathsf{P}(\square_\perp) : \mathsf{Asm}(\mathbb{N}^\mathbb{N}) \to \mathsf{Asm}(\mathbb{N}^\mathbb{N})$ by component-wise section of $\mathsf{M}\natural : \mathsf{Asm}(\mathbb{N}^\mathbb{N}) \to \mathsf{Asm}(\mathbb{N}^\mathbb{N})$ such that for each assembly $\mathbf{A}$, the underlying set of $\mathsf{P}(\mathbf{A}_\perp)$ is the underlying set of the powerdomain over the underlying set of $\mathbf{A}$. In Section 4.1.2, we define the interpretation by proving that the least fixed-points of the operators corresponding to loops are computable with regards to the moand. From the definition of the interpretation, we prove that for any ERC program, its denotation, which is a set-valued function, is computable as a multifunction in $\mathsf{Asm}(\mathbb{N}^\mathbb{N})$.

Section 4.2, we define a framework for extending ERC with other continuous data and computable operations. In Section 4.2.1, we define *extension structure* of ERC that consists of a consistent interpretation in $\mathsf{Asm}(\mathbb{N}^\mathbb{N})$ hence that we automatically get the computability result of the extended language. As examples, we define $\mathrm{ERC}(\mathcal{E}_{\mathrm{mat}})$ which is ERC extended with real matrices and $\mathrm{ERC}(\mathcal{E}_{\mathrm{rfun}})$ which is ERC extended with continuous real functions. In Section 4.2.2, we extend the verification calculus of ERC for a certain class of extension structures. In Section 4.3, we introduce Trisection program, which is a functional that finds roots of real functions, as a program in $\mathrm{ERC}(\mathcal{E}_{\mathrm{rfun}})$. We demonstrate our extended verification calculus by proving the correctness of the program.

Chapter 5 This chapter defines our second imperative language Clerical, which stands for Command-Like Expressions for Real Infinite-precision Calculations. It is a natural extension of ERC with an explicit limit operation which is an operator that returns the limit of a rapidly converging sequence. To make the limit operator useful, refraining from introducing function types, we make the expression language command-like in the sense that expressions subsume commands in Clerical. For example,

$$
\begin{aligned}
\mathsf{sqrt}(x : \mathsf{R}) :\equiv\ & \mathtt{lim}\ n.\ \mathtt{var}\ a \coloneqq \iota(0)\ \mathtt{in} \\
& \mathtt{var}\ b \coloneqq x + \iota(1)\ \mathtt{in} \\
& \mathtt{while\ case}\ b - a \mathbin{\hat{>}} 2^{-n-1} \Rightarrow \mathtt{true} \mid 2^{-n} \mathbin{\hat{>}} b - a \Rightarrow \mathtt{false\ end\ do} \\
& \quad \mathtt{var}\ c \coloneqq (\iota(2) \times a + b)/\iota(3)\ \mathtt{in} \\
& \quad \mathtt{var}\ d \coloneqq (a + \iota(2) \times b)/\iota(3)\ \mathtt{in} \\
& \quad \mathtt{case} \\
& \quad \mid \iota(0) \mathbin{\hat{>}} c \times c - x \Rightarrow a \coloneqq c \\
& \quad \mid \iota(0) \mathbin{\hat{>}} d \times d - x \Rightarrow b \coloneqq d \\
& \quad \mathtt{end} \\
& \mathtt{end}; a
\end{aligned}
$$

is an expression in Clerical that represents the square root of $x : \mathsf{R}$.

Section 5.1, we go through simple example programs in Clerical and their meanings. Hence, the readers can understand the language before going through its formal definition. In Section 5.2, the formal syntax and the type system are defined. In Section 5.3, the denotational semantics is defined. Clerical provides nondeterminism by Dijkstra style guarded nondeterminism. Clerical provides partial operations whose behaviors are different: when a comparison fails, it means nontermination $\natural$, however when computing a limit fails, it means $\sharp$. Therefore, in Section 5.3.2, we define a variant Plotkin powerdomain, distinguishing the two failures, and prove its properties, including continuity and various domain liftings that are derived from the fact that the powerdomain construction as a functor on the category of sets $\mathbb{P}_\star(\square) : \mathsf{Set} \to \mathsf{Set}$ is a moand. In Section 5.3.3, we define the denotational semantics of Clerical using the poewrdomain. In Section 5.4, we define a precondition-postcondition-style program specification and verification calculus. And, we prove that the verification calculus is sound with regards to the denotational semantics. To demonstrate the usefulness of the verification calculus, in Section 5.5.3, we prove the correctness of a program that computes π. In Section 5.6, we prove that the verification calculus is relatively complete.

Chapter 6 In this short chapter, we show that the denotational semantics of Clerical is indeed computable. We define an endofunctor $\mathsf{P}_\star(\square) : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ as a component-wise section of $\natural\mathsf{M}\natural : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ such that the definition of the endofunctor is the powerdomain construction seen as a functor $\mathbb{P}_\star(\square) : \mathsf{Set} \to \mathsf{Set}$. By showing that the domain-theoretic properties of $\mathbb{P}_\star(\square)$ are reflected in $\mathsf{P}_\star(\square)$, we show that we can construct a morphism in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ to each program in Clerical whose definition is the denotation of the program. It automatically shows that the semantics is computable.

Chapter 7 This chapter is a little off the line. Well-known algorithms in scientific computing often rely on the assumption that real numbers can be compared. However, in the setting of comptuable analysis, this is not true. Hence, in the setting, the algorithms turn out to be incorrect. One example is the QR algorithm with Wilkinson shift which is a renowned algorithm for efficiently diagonalizing symmetric matrices with real number entries. In this chapter, we suggest a computable variant of the

algorithm by nondeterministically relaxing the Wilkinson shift. In Section 7.1, we explain the chapter's own introduction, motivation, and backgrounds. In Section 7.2, we state the problem that the chapter considers formally. And, in Section 7.3, we define interval computation. In Section 7.4, we suggest the nondeterministic relaxation of the Wilkinson shift and prove its correctness. In Section 7.5 and Section 7.6, we devise computable algorithms and analyze their computational costs. It is considered an independent chapter that consists of its own introduction and conclusion.

Main Contributions

- Chapter 2: This chapter is mostly about introducing the setting that we work on. This preliminary chapter relies on [Wei00, Bau05, BHW08, VO08]. Besides other parts, Section 2.8 which states that there is no topology on the powerset that makes the notion of continuously realizable multifunctions intrinsic, is my contribution with the collaboration with Donghyun Lim [LP20]. Whereas, Specifying the functors $\flat, \sharp, \natural, \mathsf{M}$ and using them to express real computation is a minor contribution.

- Chapter 3: We define a simple imperative programming language that provides an abstract data type of real numbers based on computable analysis. That means the semantics of real number operations are exact and are computable. Hence, the language users can regard a real number variable as it is storing a real number. And, they can write a program regarding operations performing the mathematical operations of real numbers. Its formal semantics is defined using Plotkin powerdomain as the language provides nondeterminism, which is essential in computable analysis. We also devise a sound verification calculus that can be used to prove the correctness of precondition-postcondition-style program specifications. This chapter is based on the collaboration with Franz Braße, et al. [BCK+16].

- Chapter 4: We interpret ERC language in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ using the monads for partiality $\natural$ and nondeterminism M. We devise a $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ presentation of the Plotkin powerdomain. In consequence, we prove that the denotational semantics of ERC is indeed computable. By making the interpretation a framework, we suggest extending ERC with further continuous data and operations. As examples, we suggest ERC with real matrices and ERC with continuous real functions.

- Chapter 5: This chapter is based on the collaboration with Andrej Bauer and Alex Simpson [ABS18]. We devise an imperative language that extends ERC by adding explicit limit operations. Hence, real numbers can be constructed within the language. To have the limit operator, we modified the powerdomain and formalized the denotational semantics. A sound and relatively complete verification calculus is devised. An example program that computes π as the root of the sine function, which essentially has nested limit operations, is written and proved.

- Chapter 6: In this chapter, we devise a presentation of the above powerdomain in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ and prove the computability of the order completeness. In consequence, we prove the computability of the semantics of Clerical.

- Chapter 7: An interval QR algorithm with soft Wilkinson shift is suggested in this chapter; soft Wilkinson shift is a nondeterministic relaxation, which supplements partiality of the comparison in computing Wilkinson shift. An interval Gaussian algorithm is used to compute the eigenspaces. This work analyzes and specifies a condition on widths of the input intervals that guarantees the

correctness of the algorithm, returning intervals of widths less than 2^{-p}. Moreover, it analyzes the bit-cost of the computation under the condition. From the analysis of the interval computation, we show that the eigenproblem for a $d \times d$ matrix A can be computed when its entries are accessed with $\alpha \simeq \mathcal{O}(J(A,p)^2)$ and the bit-cost is bounded by $\mathcal{O}\big(d \cdot J(A,p) \cdot \mathtt{M}(J(A,p)^2)\big)$, where $J(A,p) \coloneqq d \cdot (p + d^2 + d \log 1/\Delta(A) + |\log \|A\|_F|)$, $\mathtt{M}(n)$ is the bit-cost for multiplying n-bit integers, and $\Delta(A)$ is the minimum separation between distinct eigenvalues of A relative to its Frobenius norm.

Notation and Conventions

We let $\mathbb{0}$ be the emptyset $\emptyset$, $\mathbb{1}$ be the singleton set $\{*\}$, $\mathbb{2}$ be the set of two elements $\{tt, ff\}$, $\mathbb{N}$ be the set of natural numbers, $\mathbb{Z}$ be the set of integers, $\mathbb{Q}$ be the set of rational numbers, and $\mathbb{R}$ be the set of real numbers. For any natural number $n \in \mathbb{N}$, we write $\bar{n}$ be the set $\{1, 2, \cdots, n\}$.

Given two sets X, Y, we write Y^X to denote the set of (set-theoretic) functions from X to Y. The notation $f : A \to B$ says $f \in B^A$. We often write $f : A \rightharpoonup B$ to denote that f is a *partial* function from A to B. Formally speaking, a partial function $f : A \rightharpoonup B$ is a datum that consists of a function $f : A' \to B$ and a set A such that $A' \subseteq A$. We write $\mathrm{dom}(f)$ for A'.

For a set X, we write $\mathcal{P}(X)$ for the set of subsets of X, and $\mathcal{P}_\star(X)$ for the set of nonempty subsets of X.

For a set X and an element in it $x \in X$, we write $x^{\mathbb{N}}$ to denote the infinite sequence of x; i.e., $x^{\mathbb{N}} = n \mapsto x \in X^{\mathbb{N}}$. Similarly, for a natural number n, we write x^n to denote the n-length sequence of x. If there is a possible ambiguity with it and n times repeated multiplication, it will be explicitly mentioned which the notation refers to.

For two finite sequences $x \in X^n$ and $y \in X^m$, we write $x :: y$ to denote their concatenation; i.e., $(x :: y)_k = \begin{cases} x_k & \text{if } k < n \ , \\ y_{k-n} & \text{if } k \geq n \ . \end{cases}$ When we have an infinite sequence $\varphi \in X^{\mathbb{N}}$, we write $x :: \varphi$ to denote appending x in front of φ; i.e., $(x :: \varphi)(k) = \begin{cases} x_k & \text{if } k < n \ , \\ \varphi(k-n) & \text{if } k \geq n \ . \end{cases}$ For an infinite sequence of natural numbers $\varphi \in \mathbb{N}^{\mathbb{N}}$, we write $\bar{\varphi}_n$ to denote the length n finite prefix of φ. And, $\varphi^{>}$ denotes the shifted sequence defined by $n \mapsto \varphi(n) + 1$. Similarly, $\varphi^{<}$ denotes the shifted sequence defined by $n \mapsto \varphi(n) - 1$.

For a partial function $f : A \rightharpoonup B$, we write $f \lfloor_c$ to be the total function from A to $B \cup \{c\}$ defined by

$$x \mapsto \begin{cases} f(x) & \text{if } x \in \mathrm{dom}(f) \ , \\ c & \text{otherwise} \ . \end{cases}$$

For a function $f : A \to B$, we write $f \restriction_{A'}$ to denote the domain restriction of f on $A' \subseteq A$.

When we have an expression $e(x)$ that has a free variable x, we often write $e(\Box)$ to denote the function $x \mapsto e(x)$. For example, we write $\Box^{-1}$ to denote $x \mapsto x^{-1}$. When we have an expression $e(x_1, \cdots, x_n)$ that has n free variables, we write $e(\Box_1, \cdots, \Box_n)$ to denote $(x_1, \cdots, x_n) \mapsto e(x_1, \cdots, x_n)$. For example, we write $\Box_1 + \Box_2$ for $(x, y) \mapsto x + y$.

Chapter 2. Computable Analysis

2.1 Discrete Computation

Let us start with discrete computation that we are already familiar with. Suppose we are asked to compute a function $f : A \to B$. Here, A and B are some sets, and f is a datum that connects an element of A to some element of B. To solve the problem, we need to define what it means to compute anything. Turing machine provides a formal foundation of computing over the set of natural numbers; given a Turing machine $\mathcal{M}$, its semantics is a partial function $[\![\mathcal{M}]\!] : \mathbb{N} \rightharpoonup \mathbb{N}$ where $\mathrm{dom}([\![\mathcal{M}]\!])$ is the set of inputs that make the machine terminate. Using natural numbers, which are data that machines receive and return, the sets A, B can be dealt by the machines when we number them.

A *numbered set* is a pair (A, η_A) of a set A and a partial surjective function $\eta_A : \mathbb{N} \rightharpoonup A$. For $x \in A$, if it holds that $\eta_A(n) = x$, we say n *represents* x as a datum that implements x with regards to η_A. Regarding the relation, it becomes clear what it means to compute a function $f : A \to B$ by a Turing machine: for two numbered sets (A, η_A) and (B, η_B), a Turing machine $\mathcal{M}$ *computes* a function $f : A \to B$ if the following holds for all $x \in A$:

$$\eta_B([\![\mathcal{M}]\!](n)) = f(x) \quad \text{for all } n \in \eta_A^{-1}[\{x\}].$$

The equation above implicitly requires that $[\![\mathcal{M}]\!](n)$ is well-defined. Replacing $[\![\mathcal{M}]\!]$ with any arbitrary partial function $F : \mathbb{N} \rightharpoonup \mathbb{N}$ from above, we say F *tracks* or *realizes* the function f. And, in the case there is a Turing machine $\mathcal{M}$ that computes F, i.e., $[\![\mathcal{M}]\!] = F$, we say f is computably tracked or realized by F.

The only restriction on having a set be numbered is to admit a partial surjection from $\mathbb{N}$. Hence, any countable set can be numbered. Here are some examples.

Example 2.1.

1. the emptyset $\mathbb{0}$ can be trivially numbered.
2. The singleton set $\mathbb{1} = \{*\}$ can be numbered by $\eta_{\mathbb{1}}(n) = *$ for any $n \in \mathbb{N}$.
3. The set of two elements $\mathbb{2} = \{tt, ff\}$ can be numbered by $\eta_{\mathbb{2}}$ where $\eta_{\mathbb{2}}(1) = tt$ and $\eta_{\mathbb{2}}(0) = ff$.
4. Any finite set $\bar{n}$ can be numbered by η_n where $\eta_n(k) = k \in \bar{n}$.
5. The set of natural numbers $\mathbb{N}$ can be numbered by $\eta_{\mathbb{N}}$ where $\eta_{\mathbb{N}}(n) = n$ for any $n \in \mathbb{N}$.
6. The set of integers $\mathbb{Z}$ can be numbered by $\eta_{\mathbb{Z}}$ where $\eta_{\mathbb{Z}}(2 \times k + 1) = k$ and $\eta_{\mathbb{Z}}(2 \times k) = -k$ for any $k \in \mathbb{N}$.
7. The projections $p_1, p_2 : \mathbb{N} \to \mathbb{N}$ of Cantor pairing $\langle n, m\rangle = (n + m) \cdot (n + m + 1)/2 + n$ such that $p_1(\langle n, m\rangle) = n$ and $p_2(\langle n, m\rangle) = m$ are computable. Given two computable functions $f, g : \mathbb{N} \rightharpoonup \mathbb{N}$, the function $f \times g : n \mapsto \langle f(n), g(n)\rangle$ is computable.
8. Consider any two numbered sets (A, η_A) and (B, η_B). The set-theoretic Cartesian product $A \times B \coloneqq \{(a, b) \mid a \in A \wedge b \in B\}$ can be numbered by $\eta_{A\times B} \coloneqq \langle n, m\rangle \mapsto (\eta_A(n), \eta_B(m))$. Note that the projection functions $\pi_A : A \times B \to A$ and $\pi_B : A \times B \to B$ are computable.

9. Consider any two numbered sets (A, η_A) and (B, η_B). The set-theoretic disjoint union $A + B \coloneqq \{(n, x) \mid (n = 0 \wedge x \in A) \vee (n = 1 \wedge x \in B)\}$ can be numbered by

$$\eta_{A+B}(\langle n, m\rangle) = x \quad :\Leftrightarrow \quad (n = 0 \wedge x = \eta_A(m)) \vee (n = 1 \wedge x = \eta_B(m)).$$

   Note that the injection functions $\iota_A : A \to A + B$ and $\iota_B : B \to A + B$ are computable.

10. the set of rational numbers $\mathbb{Q}$ can be numbered by $\eta_{\mathbb{Q}}(\langle n, m\rangle) = \eta_{\mathbb{Z}}(q)/\eta_{\mathbb{Z}}(m)$.

    For a rational number $q \in \mathbb{Q}$, let us write $q_{\mathbb{Q}} \in \mathbb{N}$ to denote the number of q.

11. Consider any two numbered sets (A, η_A) and (B, η_B). The set of computable functions $B^A \coloneqq \{f : A \to B \mid f \text{ is computable}\}$ can be numbered by η_{B^A} where

$$\eta_{B^A}(n) = f \quad :\Leftrightarrow \quad n\text{'th Turing machine realizes } f\ .$$

12. Most reasonable operations (whatever it means) are computable.

2.2 Type-2 Computability

When the set that we are interested in is beyond countable, it is impossible to make it numbered. For example, of real numbers, having the cardinality of Continuum, it is not possible to be represented by the set of natural numbers. Hence, classical Turing machines cannot be used. Instead, we use type-2 Turing machines for our model of computation.

Intuitively, a type-2 Turing machine is a Turing machine that reads an infinite string and writes an infinite string in its one-way infinite output tape. The output tape is one-way in the the sense that once it writes on its output tape, it cannot be modified later. The motivation is as follows. Though such a machine runs forever reading and writing infinite strings, at each time frame, it only reads a finite portion of its input and writes a finite portion of its output. And, the finite portions of its output are what the users will observe. If we allow the machine to fix its output, it is possible that what the user observes at a certain time frame is wrong in the sense that they get fixed later, such that they are not a part of the infinite output any longer. Each observation of the output, though it will never be the entire output of the machine, has to be correct finite portions of the output.

A type-2 machine's semantics is a partial function from the set of infinite sequences of natural numbers to the set of infinite sequences of natural numbers; i.e., for any type-2 machine $\mathcal{M}$, the interpretation of it is a function $[\![\mathcal{M}]\!] : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$. We call a function $F : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ *computable* if there is a type-2 machine whose interpretation is F. As the precise definition of Turing machines does not really matter (see [AB09, Chapter 1: *The computational model - and why it doesn't matter*]), we also do not need to define what type-2 Turing machines are in a precise form. Instead, here is an alternative definition of *computable functions*:

Definition 2.1.

1. An oracle Turing machine $\mathcal{M}^?$ is a Turing machine with an additional instruction $\mathsf{QUERY}(n)$. When an oracle machine $\mathcal{M}^?$ is equipped with an oracle $\varphi \in \mathbb{N}^{\mathbb{N}}$, its $\mathsf{QUERY}(n)$ with a query $n \in \mathbb{N}$ in $\mathcal{M}^\varphi$ returns $\varphi(n)$. See that it can be identified with a function of type $[\![\mathcal{M}^?]\!] : \mathbb{N}^{\mathbb{N}} \times \mathbb{N} \rightharpoonup \mathbb{N}$ where $[\![\mathcal{M}^?]\!](\varphi, n) = [\![\mathcal{M}^\varphi]\!](n)$.

2. A partial function $f : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ is *computable* if there is an oracle machine $\mathcal{M}^?$ such that $[\![\mathcal{M}^?]\!](\varphi, n) = f(\varphi)(n)$ for any $\varphi \in \mathrm{dom}(f)$ and $n \in \mathbb{N}$.

Intuitively, a function $f : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ is computable by a type-2 machine if there is an ordinary machine that can access its input $\varphi \in \mathbb{N}^{\mathbb{N}}$ by inquiring some of its entries such that when we feed an additional $n \in \mathbb{N}$, the machine prints n'th entry of $f(\varphi)$.

Unlike $\mathbb{N}$, the set $\mathbb{N}^{\mathbb{N}}$ poses interesting nontrivial structures:

Definition 2.2 (The standard topology on $\mathbb{N}^{\mathbb{N}}$ [Wei00, Definition 2.2.2])**.** The standard topology on $\mathbb{N}^{\mathbb{N}}$ is the topology generated by taking $\{a :: \mathbb{N}^{\mathbb{N}} \mid a \in \mathbb{N}^*\}$ as a subbase. Equivalently, when we define a metric $m(\varphi_1, \varphi_2) \coloneqq \inf\{2^{-n} \mid \exists a \in \mathbb{N}^n.\ \exists \varphi_1', \varphi_2' \in \mathbb{N}^{\mathbb{N}}.\ \varphi_1 = a :: \varphi_1' \wedge \varphi_2 = a :: \varphi_2'\}$, the standard topology coincides with the metric topology.

Intuitively, a function $f : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}$ is continuous if and only if $f(\varphi)$ does not depend on the infinite information of φ in the sense that there is ϵ such that for any φ' where $m(\varphi, \varphi') \leq \epsilon$, it holds that $f(\varphi) = f(\varphi')$. Similarly, a function $f : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ is continuous if and only if $f(\varphi)(n)$ depends only on some finite portion of φ. Further supposing that $f(\varphi)(n)$ is decided (whatever it means) by the finite portion of φ, the function f gets computable.

Theorem 2.1 ([Wei00, Theorem 2.2.3])**.** Any computable partial function $f : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ is continuous with regards to the standard topology on $\mathbb{N}^{\mathbb{N}}$ and the subspace topology on $\mathrm{dom}(f) \subseteq \mathbb{N}^{\mathbb{N}}$.

Let us write $\mathcal{C}(\mathbb{N}^{\mathbb{N}}, \mathbb{N}^{\mathbb{N}})$ to denote the set of continuous partial functions from $\mathbb{N}^{\mathbb{N}}$ to $\mathbb{N}^{\mathbb{N}}$ and $\mathcal{C}^{\#}(\mathbb{N}^{\mathbb{N}}, \mathbb{N}^{\mathbb{N}})$ to denote the set of computable partial functions from $\mathbb{N}^{\mathbb{N}}$ to $\mathbb{N}^{\mathbb{N}}$. The above theorem ensures that $\mathcal{C}_{\#}(\mathbb{N}^{\mathbb{N}}, \mathbb{N}^{\mathbb{N}}) \subsetneq \mathcal{C}(\mathbb{N}^{\mathbb{N}}, \mathbb{N}^{\mathbb{N}})$ holds where being a proper subset is due to a cardinality issue. There are only countably many machines whereas there are uncountably many continuous functions.

Lemma 2.1.

1. ([Wei00, Lemma 2.3.18]) For any computable $F, G : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$, the composition $G \circ F : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ is computable.

2. Consider the paring $\langle \square_1, \square_2 \rangle : \mathbb{N}^{\mathbb{N}} \times \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$ defined by the following interleaving:

$$\langle \alpha, \beta \rangle = n \mapsto \begin{cases} \alpha(k) & \text{if } n = 2k\ , \\ \beta(k) & \text{if } n = 2k + 1\ . \end{cases}$$

   The projections $p_1, p_2 : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ such that $p_1(\langle \alpha, \beta \rangle) = \alpha$ and $p_2(\langle \alpha, \beta \rangle) = \beta$ hold are computable.

3. Consider the embedding $\langle \square \rangle : (\mathbb{N}^{\mathbb{N}})^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$ defined by the Cantor pairing:

$$\langle (\varphi_n)_{n \in \mathbb{N}} \rangle = \langle n, m \rangle \mapsto \varphi_n(m)\ .$$

   The projection $p : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ such that $p(n :: \langle (\varphi_i)_{i \in \mathbb{N}} \rangle) = \varphi_n$ is computable.

And, here comes the *utm* and *smn* theorems:

Theorem 2.2 ([Wei00, § 2.2])**.**

1. There is a bijection $\boldsymbol{\eta}$ from $\mathbb{N}^{\mathbb{N}}$ to $\mathcal{C}(\mathbb{N}^{\mathbb{N}}, \mathbb{N}^{\mathbb{N}})$. Let us write $\boldsymbol{\eta}_\varphi$ to denote $\boldsymbol{\eta}(\varphi)$.

2. There is a computable partial function $\boldsymbol{u} : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ such that $\boldsymbol{u}(\langle \varphi_1, \varphi_2 \rangle) = \eta_{\varphi_1}(\varphi_2)$.

3. For any computable partial function $p : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$, there is a computable function $s : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ such that $p(\langle \alpha, \beta \rangle) = \eta_{s(\alpha)}(\beta)$.

The above theorem says there are not that many continuous functions in that continuous partial functions from $\mathbb{N}^{\mathbb{N}}$ to $\mathbb{N}^{\mathbb{N}}$ can be indexed by $\boldsymbol{\eta} : \mathbb{N}^{\mathbb{N}} \to \mathcal{C}(\mathbb{N}^{\mathbb{N}}, \mathbb{N}^{\mathbb{N}})$. And, there is an universal machine $\boldsymbol{u}$ where when we input $\langle \varphi_1, \varphi_2 \rangle$, it runs φ_2 on the φ_1'th function.

Lastly, there is an alternative characterization for computable functions.

Theorem 2.3 ([Wei00, Lemma 2.3.12])**.** Let us write $\varphi \in \mathbb{N}^{\mathbb{N}}$ be a *code* of $f : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ if $\boldsymbol{\eta}_\varphi = f$. A partial function $f : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ is computable if and only if it has a computable code.

2.3 Rep the Category of Represented Sets

2.3.1 Representations

We represent a continuous set (a set of Continuum cardinality) using a *representation*. Given a continuous set A, a representation of A is a partial surjective function $\delta_A : \mathbb{N}^{\mathbb{N}} \rightharpoonup A$. For an element $x \in A$, when an infinite sequence of natural numbers $\varphi \in \mathbb{N}^{\mathbb{N}}$ satisfies $\delta_A(\varphi) = x$, we say φ *represents* x with regards to δ_A or φ is a δ_A-*name* of x. An element $x \in A$ is δ_A-*computable* if there is a δ_A-name $\varphi \in \mathbb{N}^{\mathbb{N}}$ of x that is computable by a Turing machine; i.e., there is a Turing machine $\mathcal{M}$ where $[\![\mathcal{M}]\!] = \varphi$. We often write $\varphi \Vdash_{(A,\delta_A)} x$ for $\delta_A(\varphi) = x$.

For two represented sets (A, δ_A) and (B, δ_B), a function $f : A \to B$ is *realized* or *tracked* by $F : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ if for any $x \in A$, for each δ_A-name φ of x, it holds that $\delta_B(F(\varphi)) = f(x)$. In other words, the diagram commutes in the domain of δ_A:

$$\begin{array}{ccc} A & \xrightarrow{\quad f \quad} & B \\ {\scriptstyle \delta_A}\uparrow & & \uparrow{\scriptstyle \delta_B} \\ \mathbb{N}^{\mathbb{N}} & \xrightarrow[\quad F \quad]{} & \mathbb{N}^{\mathbb{N}} \end{array}$$

In the case, we write $F \Vdash_{(A,\delta_A)\to(B,\delta_B)} f$. When F is computable, we say f is (δ_A, δ_B)-*computably realized* by F. And, when F is continuous, we say f is (δ_A, δ_B)-*continuously realized* by F. When it is obvious from the context which representations underlie, we often omit the prefix (δ_A, δ_B)-. For a represented set $\mathbf{A}$, we write $|\mathbf{A}|$ to refer to the underlying set of $\mathbf{A}$ and $\delta_{\mathbf{A}}$ to refer to the representation of $\mathbf{A}$. When there is no possible ambiguity, we often simply write A and δ_A to refer to $|\mathbf{A}|$ and $\delta_{\mathbf{A}}$.

Example 2.2.

1. Any numbered set $\mathbf{A} = (A, \eta_A)$ can be represented by $\delta_A : \mathbb{N}^{\mathbb{N}} \rightharpoonup A$ where

$$\delta_A(\varphi) = \eta_A(\varphi(0)) \, .$$

   We call a representation on a countable set A *standard* if there is a numbered set that generates the representation in the above way. Let us write $\mathbf{0}$, $\mathbf{1}$, $\mathbf{2}$, $\mathbf{N}$, $\mathbf{Z}$, and $\mathbf{Q}$ to be the represented sets of $\mathbb{0}$, $\mathbb{1}$, $\mathbb{2}$, $\mathbb{N}$, $\mathbb{Z}$, and $\mathbb{Q}$ that are generated from the numberings in Example 2.1.

   Computable functions in the setting of numbering are computable in the setting of represented sets.

2. We call a function $\delta_{\mathrm{Cauchy}} : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{R}$ such that

$$\delta_{\mathrm{Cauchy}}(\varphi) = x \quad :\Leftrightarrow \quad |x - \eta_{\mathbb{Q}}(\varphi(n))| \leq 2^{-n} \text{ for all } n \in \mathbb{N}$$

the standard representation of real numbers. In words, φ represents a real number x when it encodes a sequence of rationals that *rapidly* converges to x; a sequence of real numbers $(x_i)_{i\in\mathbb{N}} \subseteq \mathbb{R}$ is said to converge rapidly if there is $x \in \mathbb{N}$ such that $|x - x_n| \leq 2^{-n}$ holds for all $n \in \mathbb{N}$. Let us write $\mathbf{R}_{\mathrm{Cauchy}}$ for $(\mathbb{R}, \delta_{\mathrm{Cauchy}})$.

3. For any represented set $\mathbf{A} = (A, \delta_A)$, the identity function $\mathrm{id}_A : A \to A$ is trivially computable.

4. For any three represented sets $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and continuously (computably) realizable functions $f : A \to B$, $g : B \to C$, their composition $g \circ f : A \to C$ is continuously (computably) realizable.

5. For two represented sets $\mathbf{A} = (A, \delta_A)$ and $\mathbf{B} = (B, \delta_B)$, the set-theoretic Cartesian product $A \times B$ can be represented by $\delta_{A\times B}$ that is defined as follows:

$$\delta_{A\times B}\big(\langle \varphi_1, \varphi_2\rangle\big) = (x, y) \quad :\Leftrightarrow \quad \varphi_1 \Vdash_{\mathbf{A}} x \wedge \varphi_2 \vdash_{\mathbf{B}} y\ .$$

Let us write $\mathbf{A}\times\mathbf{B}$ be the represented set $(A\times B, \delta_{A\times B})$. Let us call $\delta_{A\times B}$ *the product representation* of δ_A and δ_B. The projections $\pi_A : A \times B \to A$ and $\pi_B : A \times B \to B$ are computable.

6. For represented sets $\mathbf{A}, \mathbf{B}, \mathbf{X}$ and continuously (computably) realizable $f : X \to A$ and $g : X \to B$, the map $f \times g : X \ni x \mapsto (f(x), g(x)) \in A \times B$ is continuously (computably) realizable.

7. For two represented sets $\mathbf{A} = (A, \delta_A)$ and $\mathbf{B} = (B, \delta_B)$, the set-theoretic disjoint union $A + B = (\{0\} \times A) \cup (\{1\} \times B)$ can be represented by δ_{A+B} that is defined as follows:

$$\delta_{A+B}\big(n :: \varphi\big) = (n, x) \quad :\Leftrightarrow \quad (n = 0 \wedge \varphi \Vdash_{\mathbf{A}} x) \vee (n = 1 \wedge \varphi \Vdash_{\mathbf{B}} x)\ .$$

Let us write $\mathbf{A}+\mathbf{B}$ be the represented set $(A+B, \delta_{A+B})$ and call δ_{A+B} the coproduct representation of δ_A and δ_B. The injections $\iota_A : A \to A + B$ and $\iota_B : B \to A + B$ are computable.

8. For represented sets $\mathbf{A}, \mathbf{B}, \mathbf{X}$ and continuously (computably) realizable $f : A \to X$ and $g : B \to X$, the map

$$f + g : (A + B) \ni x \mapsto \begin{cases} f(y) & \text{if } x = \iota_A(y), \\ g(y) & \text{if } x = \iota_B(y), \end{cases}$$

is continuously (computably) realizable.

9. For represented sets $\mathbf{A}$, the conditional function $\mathsf{Cond}_{\mathbf{A}} : |\mathbf{2} \times \mathbf{A} \times \mathbf{A}| \to |\mathbf{A}|$ defined by

$$\mathsf{Cond}_{\mathbf{A}}(tt, x, y) = x \text{ and } \mathsf{Cond}_A(\mathit{ff}, x, y) = y$$

is computable.

10. For a represented set $\mathbf{A} = (A, \delta_A)$ and a subset $B \subseteq A$, a representation δ_B of B is a *subrepresentation* of δ_A when it is defined by

$$\delta_B(\varphi) = x \quad :\Leftrightarrow \quad \delta_A(\varphi) = x \quad \text{for all } x \in B.$$

Let us write $(B, \delta_B) \subseteq \mathbf{A}$ to denote that δ_B is a subrepresentation of δ_A. And, let us write $\mathsf{sub}_B(\mathbf{A})$ to denote the represented set (B, δ_B) where $(B, \delta_B) \subseteq \mathbf{A}$. See that the subset inclusion $\mathsf{sub}_B(\mathbf{A}) \rightarrowtail \mathbf{B}$ is realized by $\mathrm{id} : \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$.

11. More generally, for a represented set $\mathbf{B}$, a set A, and an injective function $\iota : A \to |\mathbf{B}|$, the *subrepresented set induced* by $\iota : A \to |\mathbf{B}|$ is a represented set $\mathbf{A}$ on A defined by

$$\varphi \Vdash_{\mathbf{A}} x \quad :\Leftrightarrow \quad \varphi \Vdash_{\mathbf{B}} \iota(x).$$

See that the injective function $\iota : |\mathbf{A}| \to |\mathbf{B}|$ is computable.

12. For two represented sets $\mathbf{A} = (A, \delta_A)$ and $\mathbf{B} = (B, \delta_B)$, the set of continuously realizable functions $\mathcal{C}^{\downarrow}(\mathbf{A}, \mathbf{B})$ can be represented by δ_{B^A} which is defined as follows:

$$\varphi \Vdash_{\delta_{B^A}} f \quad :\Leftrightarrow \quad \boldsymbol{\eta}_{\varphi} \Vdash_{\mathbf{A}\to\mathbf{B}} f\ .$$

In words, φ represents f if φ is a code of f. Let us write $\mathbf{B}^{\mathbf{A}}$ be the represented set $(\mathcal{C}^{\downarrow}(\mathbf{A}, \mathbf{B}), \delta_{B^A})$ and call δ_{B^A} *the function representaion* of δ_A and δ_B. The evaluation map $\mathsf{eval}_{A,B} : \mathcal{C}^{\downarrow}(\mathbf{A}, \mathbf{B}) \times A \to B$ is computable.

13. For any represented sets $\mathbf{A}, \mathbf{B}, \mathbf{C}$, and a computable (continuously realizable) function $f : |\mathbf{C}\times\mathbf{A}| \to |\mathbf{C}|$, its transpose $\bar{f} : |\mathbf{C}| \to |\mathbf{B}^{\mathbf{A}}|$ is computable (continuously realizable).

14. For a represented set $\mathbf{A} = (A, \delta_A)$, define $\mathsf{Seq}(\mathbf{A}) = (A^{\mathbb{N}}, \delta)$ be a represented set of infinite sequences based on the Cantor pairing:

$$\delta\big(\langle(\varphi_i)_{i\in\mathbb{N}}\rangle\big) = (x_i)_{i\in\mathbb{N}} \quad :\Leftrightarrow \quad \varphi_i \Vdash_{\mathbf{A}} x_i \text{ for all } i \in \mathbb{N}\ .$$

The entry access function $\mathsf{access}_A : A^{\mathbb{N}} \times \mathbb{N} \to A$ is computable with regards to the standard representation of $\mathbb{N}$.

See that any infinite sequence $f : \mathbb{N} \to A$ is continuously realizable. Hence, the set of infinite sequences of A coincides with $\mathcal{C}^{\downarrow}(\mathbf{N}, \mathbf{A})$.

The above examples suggest that when we collect represented sets and continuously (computable) realizable functions, they form a very well structured category. Let us write Rep to be the *category of represented sets with computable functions as morphisms*. And, $\mathsf{Rep}_{\mathsf{cont}}$ be the *category of represented sets with continuously realizable functions as morphisms*. In this dissertation, we are mostly interested in Rep, and otherwise it is explicitly mentioned, we refer to Rep by saying the category of represented sets.

The category Rep is Cartesian closed with $\mathbf{1}$ being a terminal object, $\mathbf{0}$ being the initial object, $\mathbf{A}\times\mathbf{B}$ from Example 2.2 (5) being a product of $\mathbf{A}$ and $\mathbf{B}$, $\mathbf{A}+\mathbf{B}$ from Example 2.2 (7) being a coproduct of $\mathbf{A}$ and $\mathbf{B}$, and $\mathbf{B}^{\mathbf{A}}$ from Example 2.2 (12) being an exponential object of $\mathbf{A}$ and $\mathbf{B}$.

Note that even when we take only computable functions as morphisms, the underlying set of an exponent is the set of continuously realizable functions. Hence, a realizer of a morphism $\mathbf{B}^{\mathbf{A}} \to \mathbf{C}$ must not assume that a name φ of a function $f \in |\mathbf{B}^{\mathbf{A}}|$ is computable.

We often write $\mathbf{A} \to \mathbf{B}$ instead of $\mathbf{B}^{\mathbf{A}}$. The symbols $\times, +$ seen as operators are left-associative, meanwhile $\to$ is right-associative. That means, $\mathbf{A} \to \mathbf{B} \to \mathbf{C}$ denotes $\mathbf{A} \to (\mathbf{B} \to \mathbf{C})$, $\mathbf{A}\times\mathbf{B}\times\mathbf{C}$ denotes $(\mathbf{A}\times\mathbf{B})\times\mathbf{C}$, and $\mathbf{A}+\mathbf{B}+\mathbf{C}$ denotes $(\mathbf{A}+\mathbf{B})+\mathbf{C}$. The precedence of the symbol $\times$ is the highest and the precedence of the symbol $\to$ is the lowest amongst the three. That is, we parse $\mathbf{A}+\mathbf{B} \to \mathbf{B}\times\mathbf{C}+\mathbf{B}$ as $(\mathbf{A}+\mathbf{B}) \to ((\mathbf{B}\times\mathbf{C})+\mathbf{B})$.

Putting represented sets into the framework of category theory, it provides a natural way to define relations between represented sets. Note that a set can admit many different representations. For

example, the underlying sets of $\mathsf{Seq}(\mathbf{A})$ and $\mathbf{N} \to \mathbf{A}$ are identical as the set of infinite sequences of elements in A. However, they are represented in different ways. And, this leads to a very natural question: which one is right to use?

Two represented sets $\mathbf{A} = (A, \delta_A)$ and $\mathbf{B} = (B, \delta_B)$ are *equivalent* $\mathbf{A} \cong \mathbf{B}$ if they are isomorphic in Rep: i.e., $\mathbf{A}$ and $\mathbf{B}$ are equivalent if there are two computable functions $f : A \to B$ and $g : B \to A$ where $f \circ g = \mathrm{id}_A$ and $g \circ f = \mathrm{id}_B$ hold. Being equivalent is justified that when $\mathbf{A} \cong \mathbf{B}$, there is a computable translation back and forth. Hence, a computable function $f : \mathbf{B} \to \mathbf{C}$ can be automatically transformed to a function $g : \mathbf{A} \to \mathbf{C}$. For example, we can easily see that $\mathbf{2} \cong \mathbf{1} + \mathbf{1}$. The following lemma states that it does not matter in which specific way the set of sequences is represented:

Lemma 2.2. For any represented set $\mathbf{A}$, the represented sets $\mathbf{N} \to \mathbf{A}$ and $\mathsf{Seq}(\mathbf{A})$ are equivalent.

In Example 2.2 (2), we defined a representation of real numbers and named it *the standard representation of real numbers*. The name suggests that the representation is the representation that we are particularly interested in. Let us conclude this subsection revealing some (but not all) properties of the representation.

Example 2.3 (Operations using $\mathbf{R}_{\mathrm{Cauchy}}$)**.** The constants $0, 1 \in \mathbb{R}$, the real number addition $\square_1 + \square_2 : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, the real number subtraction $\square_1 - \square_2 : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, and the real number multiplication $\square_1 \times \square_2 : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ are computable.

However, the order relations $\square_1 < \square_2, \square_1 > \square_2 : \mathbb{R} \times \mathbb{R} \to 2$ and the identity relation $\square_1 = \square_2 : \mathbb{R} \times \mathbb{R} \to 2$ are not computable.

Proof. The computability results can be easily shown. See [Wei00, Theorem 4.3.2] for a reference.

For the noncomputability result, see that if $\square_1 < \square_2$ or $\square_1 > \square_2$ was computable, we can have $\texttt{if } x < y \texttt{ then false else if } y < x \texttt{ then false else true}$ to compute $x = y$. Hence, we only need to show that $\square_1 = \square_2$ is not computable. Suppose there is a continuous realizer $\tau : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ of $\square_1 = \square_2 : \mathbb{R} \times \mathbb{R} \to 2$. See that $(0_{\mathbb{Q}})^{\mathbb{N}}$ is a name of $0 \in |\mathbf{R}_{\mathrm{Cauchy}}|$. Hence, $\tau(\langle (0_{\mathbb{Q}})^{\mathbb{N}}, (0_{\mathbb{Q}})^{\mathbb{N}} \rangle) = 0 :: \varphi$ for some $\varphi \in \mathbb{N}^{\mathbb{N}}$ which is a name of $tt \in |\mathbf{2}|$. Since τ is continuous, there is $m \in \mathbb{N}$ where for any $\langle \alpha, \beta \rangle$ such that $d(\langle \alpha, \beta \rangle, \langle (0_{\mathbb{Q}})^{\mathbb{N}}, (0_{\mathbb{Q}})^{\mathbb{N}} \rangle) \leq 2^{-m}$, it holds that $\tau(\langle \alpha, \beta \rangle)(0) = 0$. However, when we define $\alpha := n \mapsto \begin{cases} 0_{\mathbb{Q}} & \text{if } n < m \\ (2^{-m-1})_{\mathbb{Q}} & \text{otherwise.} \end{cases}$, which is a name of 2^{-m-1}, since $2^{-m-1} \neq 0$, it has to be that $\tau(\langle \alpha, (0_{\mathbb{Q}})^{\mathbb{N}} \rangle)(0) = 1$. It is a contradiction since by the definition of the interleaving, $d(\langle \alpha, (0_{\mathbb{Q}})^{\mathbb{N}} \rangle, \langle (0_{\mathbb{Q}})^{\mathbb{N}}, (0_{\mathbb{Q}})^{\mathbb{N}} \rangle) \leq 2^{-m}$ holds. □

Remark 2.1. The category of represented sets being a Cartesian closed category, we can use its internal language, which is a typed lambda calculus, to construct various objects in the category: represented sets and computable functions. There are two different approaches to finding a computable function. The first classical way is to specify a set-theoretic function that we are interested in and argue its computability. The second, more preferable approach is to use the internal language to construct computable functions at once.

It is a typed lambda calculus where represented sets are types and morphisms are terms. The biggest advantage is that we can use variables for defining morphisms. First, note that the function $(\mathsf{half} : |\mathbf{R}_{\mathrm{Cauchy}}| \to |\mathbf{R}_{\mathrm{Cauchy}}|) := x \mapsto x/2$ is computable. Using it, we simply write

$$2^{-\square} := \mathsf{nat\_rec}_{\mathbf{R}_{\mathrm{Cauchy}}}(1, \lambda(m : \mathsf{N}).\ \lambda(x : \mathbf{R}_{\mathrm{Cauchy}}).\ \mathsf{half}(x))$$

to define the mapping $\mathbb{N} \ni n \mapsto 2^{-n} \in \mathbb{R}$. Here, $\mathsf{nat\_rec}_{\mathbf{A}}$ for a represented set $\mathbf{A}$ is a primitive recursor

$$\mathsf{nat\_rec}_{\mathbf{A}} : \mathbf{A} \to (\mathbf{N} \to \mathbf{A} \to \mathbf{A}) \to \mathbf{N} \to \mathbf{A}$$

that is defined by

$$\mathsf{nat\_rec}_{\mathbf{A}}\ x\ f\ 0 = x \quad \text{and} \quad \mathsf{nat\_rec}_{\mathbf{A}}\ x\ f\ (n+1) = f\ n\ (\mathsf{nat\_rec}_{\mathbf{A}}\ x\ f\ n)$$

where its computability is left as an exercise.

Nested lambda expression $\lambda(x_1 : \mathbf{A}_1).\ \cdots\ \lambda(x_n : \mathbf{A}_n).\ t(x_1, \cdots, x_n)$ denotes a morphism of type $\mathbf{A}_1 \to \mathbf{A}_2 \to \cdots \mathbf{A}_n \to \mathbf{B}$. For convenience in the presentation, we often convert it to a n-ary function $\mathbf{A}_1 \times \cdots \times \mathbf{A}_n \to \mathbf{B}$ by implicitly uncurrying it.

2.3.2 Partial Functions

Thus far, only total functions were subject to being computed or realized. Though underlying representations and realizers were allowed to be partial, functions that to be realized had to be total. However, it is not always the case that we are interested in computing total functions. For example, the multiplicative inversion of real numbers, which we obviously want to compute, is a partial function that is not defined at zero.

An obvious choice of the definition for realizing a partial function would be as follows. Given two represented sets $\mathbf{A}$ and $\mathbf{B}$, a partial function $f : A \rightharpoonup B$ is realized by $F : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ if F realizes $f : \mathrm{dom}(f) \to B$ with regards to $\mathsf{sub}_{\mathrm{dom}(f)}(\mathbf{A})$ and $\mathbf{B}$. And, say f is computable (continuously realizable) if there is a computable (continuously realizable) realizer.

A downside of this definition is that for two represented sets $\mathbf{A}$ and $\mathbf{B}$, there is no represented set, in general, for the set of continuously realizable partial functions from $\mathbf{A}$ to $\mathbf{B}$. Suppose F realizes a function $f : A \to B$. Then, for each subset S of A, its restriction $f\restriction_S : A \rightharpoonup B$ is realized by F. Since any constant function can be continuously realized, the cardinality of the set of continuously realizable partial functions is at least the cardinality of the powerset of A. Hence, when A is continuous, the set of continuously realizable partial functions from A to B does not admit any serjection from $\mathbb{N}^{\mathbb{N}}$.

Hence, in order to refer to a computable or continuously realizable partial function in Rep, we need to specify the domain. That is, we cannot quantify over all continuously realizable partial functions from a represented set to another represented set. (We cannot have $\lambda(f : \mathbf{A} \rightharpoonup \mathbf{B}).t(f)$ for example.)

The reason is simply that there are too many of them. A realizer F of a partial function $f : A \rightharpoonup B$ is allowed to do anything when a name φ of an element of A not in $\mathrm{dom}(f)$ is given. That means, $F(\varphi)$ can either diverge (i.e., $\varphi \notin \mathrm{dom}(F)$) or return anything (i.e., $\varphi \in \mathrm{dom}(F)$ but $F(\varphi)$ does not need to mean anything).

We can refine the set of continuously realizable functions by forcing the behaviour of their realizers on wrong inputs as in classical computability theory. A partial function $f : \mathbf{A} \rightharpoonup \mathbf{B}$ is *strongly realized* by $F : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ if for any $x \in A$, when $x \notin \mathrm{dom}(f)$, for any φ such that $\varphi \Vdash_{\mathbf{A}} x$, it holds that $\varphi \notin \mathrm{dom}(F)$.

Definition 2.3. For two represented sets $\mathbf{A}$ and $\mathbf{B}$, define $\mathbf{C}^{\flat}(\mathbf{A}, \mathbf{B})$ be a represented set of the set of continuously and strongly realizable partial functions from $|\mathbf{A}|$ to $|\mathbf{B}|$ whose representation is as follows.

$$\varphi \Vdash_{\mathbf{C}^{\flat}(\mathbf{A},\mathbf{B})} f \quad :\Leftrightarrow \quad (\boldsymbol{\eta}_{\varphi} \Vdash_{\mathsf{sub}_{\mathrm{dom}(f)}(\mathbf{A}) \to \mathbf{B}} f) \wedge \mathrm{dom}(\boldsymbol{\eta}_{\varphi}) = \delta_{\mathbf{A}}^{-1}(\mathrm{dom}(f))$$

See that, by definition, each $f \in |\mathbf{C}^{\flat}(\mathbf{A}, \mathbf{B})|$ has a name. And, each φ can represent at most one $f \in |\mathbf{C}^{\flat}(\mathbf{A}, \mathbf{B})|$.

We see if the definition is effective in that if there is a represented set $\mathbf{B}'$ where $\mathbf{A} \to \mathbf{B}'$ is equivalent to the represented set of strongly and continuously realizable partial functions from $|\mathbf{A}|$ to $|\mathbf{B}|$.

Definition 2.4 (Lazy Lifting)**.** Lazy lifting is an endofunctor $\flat : \mathsf{Rep} \to \mathsf{Rep}$. The functor on a represented set $\mathbf{A} = (A, \delta_A)$ is $\flat\mathbf{A}$ whose underlying set is $A \cup \{\flat_{\mathbf{A}}\}$ and representation $\delta_{\flat\mathbf{A}}$ is

$$0^{\mathbb{N}} \Vdash_{\flat\mathbf{A}} \flat_{\mathbf{A}} \text{ and } 0^m :: \varphi^{>} \Vdash_{\delta_{\flat\mathbf{A}}} x \quad :\Leftrightarrow \quad \varphi \Vdash_{\mathbf{A}} x \text{ for any } m \in \mathbb{N}.$$

In words, only the infinite sequence of zeros realizes $\flat_{\mathbf{A}}$. And, a shifted sequence $\varphi^{>}$ with a prefix of finitely many zeros realizes what φ realizes in $\mathbf{A}$.

The functor on a morphism $f : \mathbf{A} \to \mathbf{B}$ is defined to be

$$\flat(f) : x \mapsto \begin{cases} f(x) & \text{if } x \neq \flat_{\mathbf{A}} \ , \\ \flat_{\mathbf{B}} & \text{otherwise} \ . \end{cases}$$

When F is a computable realizer of f, see that the following procedure computes $\flat(f)$. Suppose φ is a name of some $x \in \flat\mathbf{A}$. Repeat the following with increasing n by 1 starting from $n = 0$.

1. when $\varphi(n) = 0$, append 0 in the output tape
2. when $\varphi(n) \neq 0$, i.e., when $\varphi = 0^{n-1} :: \psi^{>}$ for some ψ, append $F(\psi)^{>}$ in the output tape.

In other words, append 0 until we find a nonzero element from φ. If there is a nonzero entry in φ, it means x is not $\flat_{\mathbf{A}}$, and its original name is the infinite string starting from the entry shifted back. Hence, in this case, apply F on the name, shift the result, and print it on the output tape.

See that the above procedure only requires F to be continuous and is computable when F is seen as an input. □

Let us remove the subscript $\mathbf{A}$ from $\flat_{\mathbf{A}}$ if it is clear from the context or is irrelevant which represented set the added element belongs to. Subscript is needed to be explicitly written only when we lift a represented set twice thus that there are two added elements $\flat_{\mathbf{A}}$ and $\flat_{\flat\mathbf{A}}$ that are distinct.

For a represented set, $\flat$ in its lazy lifting denotes nontermination. See that for any φ, we cannot, in a finite time, decide if φ represents $\flat$ or not; only infinitely many consecutive zeros represent $\flat$. However, after reading a finite portion, it is possible that consecutive zeros end. In this case, the sequence will realize something different from $\flat$.

For a partial function $f : A \rightharpoonup B$, let us say $f \lfloor_\flat : A \to B \cup \{\flat\}$ the *lazy extension* of f.

Lemma 2.3. Consider represented sets $\mathbf{A}, \mathbf{B}$. A partial function $f : |\mathbf{A}| \rightharpoonup |\mathbf{B}|$ is continuously (computably) and strongly realizable if and only if its lazy extension $(f \lfloor_\flat) : |\mathbf{A}| \to |\flat\mathbf{B}|$ is continuously (computably) realizable. More specifically, the mapping $(f \mapsto f \lfloor_\flat) : \mathbf{C}^\flat(\mathbf{A}, \mathbf{B}) \to (\mathbf{A} \to \flat\mathbf{B})$ is an isomorphism.

We can refer to the set of continuously realizable partial functions by $(\mathbf{A} \to \flat\mathbf{B})$ and a strongly computable partial function by $f : \mathbf{A} \to \flat\mathbf{B}$.

After characterizing an important class of partial functions, we can think of its dual notion.

Definition 2.5 (Co-lazy lifting)**.** Co-lazy lifting is an endofunctor $\sharp : \mathsf{Rep} \to \mathsf{Rep}$. The functor on a represented set $\mathbf{A} = (A, \delta_{\mathbf{A}})$ is $\sharp\mathbf{A}$ whose underlying set is $A \cup \{\sharp_{\mathbf{A}}\}$ and representation $\delta_{\sharp\mathbf{A}}$ is

$$\begin{aligned} \varphi \Vdash_{\sharp\mathbf{A}} \sharp_{\mathbf{A}} \quad &:\Leftrightarrow \quad \exists n.\, \varphi(n) = 0 \\ \varphi^{>} \Vdash_{\sharp\mathbf{A}} x \quad &:\Leftrightarrow \quad \varphi \Vdash_{\mathbf{A}} x \ . \end{aligned}$$

In words, any sequence containing 0 represents the special element $\sharp_{\mathbf{A}}$. And, φ represents $x \neq \sharp_{\mathbf{A}}$ if and only if it does not contain 0 and when we shift it by -1, it represents x in $\mathbf{A}$.

The functor on a morphism $f : \mathbf{A} \to \mathbf{B}$ is defined by

$$\sharp(f) : x \mapsto \begin{cases} f(x) & \text{if } x \neq \sharp_{\mathbf{A}} \ , \\ \sharp_{\mathbf{B}} & \text{otherwise.} \end{cases}$$

When F is a computable realizer of f, and φ is a name of an input, see that computing $F(\varphi^{<})^{>}$ while searching for 0 in φ and append 0 if there is, computes $\sharp(f)$. Also, this procedure extends to all continuously realizable functions and is computable regarding F as an input.

Similarly, we drop the subscript from $\sharp$ if it is clear from the context or is irrelevant.

As $\flat$ classified a class of partial functions, $\sharp$ as well classifies another class of partial functions.

Definition 2.6. A partial function $f : |\mathbf{A}| \rightharpoonup |\mathbf{B}|$ is weakly and continuously (computably) realizable if $f \lfloor_\sharp$ as a function from $|\mathbf{A}|$ to $|\sharp\mathbf{B}|$ continuously (computably) realizable.

See that, by definition, any partial function that is weakly or strongly computable is computable.

Remark 2.2. A partial function is strongly computable if and only if its domain is semi-decidable and is weakly computable if and only if its domain is co-semi-decidable. A partial function is both strongly and weakly computable if and only if its domain is a decidable subset.

We defined the various notions of computing partial functions because we need them to analyze some essential partial functions.

Example 2.4. Consider $\mathbf{2}, \mathbf{N}$ and $\mathbf{R}_{\text{Cauchy}}$.

1. The multiplicative inversion as a partial function $\square^{-1} : \mathbb{R} \rightharpoonup \mathbb{R}$ that is not defined at 0 is strongly computable. Hence, its lazy extension $\square^{-1}\lfloor_\flat$ is a morphism from $\mathbf{R}_{\text{Cauchy}}$ to $\flat\mathbf{R}_{\text{Cauchy}}$.

2. Define $\square_1 < \square_2 : \mathbb{R} \times \mathbb{R} \rightharpoonup 2$ be a *partial* approximation of the real number comparison test such that
$$x < y = \begin{cases} tt & \text{if } x < y \ , \\ ff & \text{if } x > y \ . \end{cases}$$
See that $\text{dom}(\square_1 < \square_2) = \{(x, y) \in \mathbb{R}^2 \mid x \neq y\}$. The partial function is *strongly* computable. In other words, its lazy extension $\square_1 < \lfloor_\flat \ \square_2$ is a morhpism from $\mathbf{R}_{\text{Cauchy}} \times \mathbf{R}_{\text{Cauchy}}$ to $\flat\mathbf{2}$.

3. The limit operation $\lim : \mathbb{R}^{\mathbb{N}} \rightharpoonup \mathbb{R}$ as a partial function such that $\text{dom}(\lim) = \{(x_i)_{i\in\mathbb{N}} \in \mathbb{R}^{\mathbb{N}} \mid (x_i)_{i\in\mathbb{N}} \text{ converges}\}$ is not computable.

4. Instead, define a refinement $\mathsf{lim} : \mathbb{R}^{\mathbb{N}} \rightharpoonup \mathbb{R}$ by restricting the domain to $\{(x_i)_{i\in\mathbb{N}} \in \mathbb{R}^{\mathbb{N}} \mid \exists z.\ z \in \mathbb{R} \wedge |z - x_i| \leq 2^{-n}\}$. In words, lim is defined only at rapidly converging sequences. Then, the partial function $\mathsf{lim} : \mathbb{R}^{\mathbb{N}} \rightharpoonup \mathbb{R}$ is weakly computable. That means, its colazy extension $\mathsf{lim}\lfloor_\sharp$ is a moprhism from $\mathbf{N} \to \mathbf{R}_{\text{Cauchy}}$ to $\sharp\mathbf{R}_{\text{Cauchy}}$.

2.4 Applicative Functors and Monads

Endofunctors perform as type constructors in the internal language of a category. For example, when $\mathbf{A}$ is seen as a data type, $\flat\mathbf{A}$ is another data type that is deeply relevant to $\mathbf{A}$. It is deeply relevant in the sense that for a morphism $f : \mathbf{A} \to \mathbf{B}$, we automatically get a computable mapping $\flat(f) : \flat\mathbf{A} \to \flat\mathbf{B}$.

For example, instead of defining a function $\neg'\Box : \flat\mathbf{2} \to \flat\mathbf{2}$ that maps tt to ff, ff to tt, and $\flat$ to $\flat$, then proving its computability, we can extend $\flat(\neg\Box : \mathbf{2} \to \mathbf{2}) : \flat\mathbf{2} \to \flat\mathbf{2}$ which is exactly the desired function. And, the computability comes free.

However, it is not that straightforward when the function we want to lift accepts multiple arguments. Let us see three examples that require different ways of lifting.

$$\Box_1 + \Box_2 : \mathbf{R}_{\text{Cauchy}} \times \mathbf{R}_{\text{Cauchy}} \to \mathbf{R}_{\text{Cauchy}}, \quad \mathsf{Cond}_{\mathbf{A}} : \mathbf{2} \times \mathbf{A} \times \mathbf{A} \to \mathbf{A}, \quad \text{and} \quad \Box^{-1\lfloor\flat} : \mathbf{R}_{\text{Cauchy}} \to \flat\mathbf{R}_{\text{Cauchy}}.$$

Consider the first example, $\Box_1 + \Box_2 : \mathbf{R}_{\text{Cauchy}} \times \mathbf{R}_{\text{Cauchy}} \to \mathbf{R}_{\text{Cauchy}}$. Suppose by some computation involving partialities that we get lazy real numbers $x, y : \flat\mathbf{R}_{\text{Cauchy}}$. We are not sure whether x, y are $\flat$ or not. However, without revealing those, we can think of adding the two lazy real numbers where the result is also a lazy real number that happens to be $\flat$ when x or y is $\flat$. Formalizing it, we desire the function

$$(x, y) \mapsto \begin{cases} x + y & \text{if } x \neq \flat \wedge y \neq \flat, \\ \flat & \text{otherwise.} \end{cases}$$

See that this is of course not $\flat(\Box_1 + \Box_2) : \flat(\mathbf{R}_{\text{Cauchy}} \times \mathbf{R}_{\text{Cauchy}}) \to \flat\mathbf{R}_{\text{Cauchy}}$. This example illustrates the case where we want $\flat$ in the domain distributes through its products.

For the second example, $\mathsf{Cond}_{\mathbf{A}} : \mathbf{2} \times \mathbf{A} \times \mathbf{A} \to \mathbf{A}$ the conditional, suppose we have a lifted Boolean $b : \flat\mathbf{2}$ obtained by comparing two real numbers. We can think of branching according to b such that when $b = tt$, we take the first branch, when $b = ff$, we take the second branch, and when $b = \flat$, we simply return $\flat$. The version of lifting we require is

$$(b, x, y) \mapsto \begin{cases} x & \text{if } b = tt, \\ y & \text{if } b = ff, \\ \flat & \text{if } b = \flat. \end{cases}$$

That means, we only want to lift the first argument of $\mathsf{Cond}_{\mathbf{A}} : \mathbf{2} \times \mathbf{A} \times \mathbf{A} \to \mathbf{A}$, whereas $\flat(\mathsf{Cond}_{\mathbf{A}}) : \flat(\mathbf{2} \times \mathbf{A} \times \mathbf{A}) \to \flat\mathbf{A}$.

The last example, $\Box^{-1\lfloor\flat} : \mathbf{R}_{\text{Cauchy}} \to \flat\mathbf{R}_{\text{Cauchy}}$ is when we want to lift a mapping whose codomain is already lifted. When we simply lift it $\flat\big(\Box^{-1\lfloor\flat}\big)$, its codomain is $\flat\flat\mathbf{R}_{\text{Cauchy}}$ that $0^{\flat(-1\lfloor\flat)} = \flat_{\mathbf{R}_{\text{Cauchy}}}$ and $\flat^{\flat(-1\lfloor\flat)}_{\mathbf{R}_{\text{Cauchy}}} = \flat_{\flat\mathbf{R}_{\text{Cauchy}}}$. However, since $\flat$ denotes nontermination, there is no need to have two distinct $\flat$s in the codomain.

We can, of course, define the liftings for each endofunctor on an ad hoc. For the case of $\flat : \mathsf{Rep} \to \mathsf{Rep}$, it is obvious how to define those. However, there is a remaining question if there is a natural and uniform way to define the liftings and if the above examples follow the standard approach.

Consider any category C with a terminal object $\mathbf{1}$ and every products[1]. An endofunctor $F : \mathsf{C} \to \mathsf{C}$ is *lax monoidal* if it is equipped with a morphism $\varepsilon : \mathbf{1} \to F(\mathbf{1})$ and a natural transformation

$$\alpha_{\mathbf{A},\mathbf{B}} : F(\mathbf{A}) \times F(\mathbf{B}) \to F(\mathbf{A} \times \mathbf{B})$$

[1] To be precise, C needs to be monoidal. However, the categories used in this dissertation have a terminal and products, and we use only the monoidal structures of the categories based on those.

such that the coherence conditions defined by the following three diagrams are satisfied:

$$\begin{array}{ccc}
(F(\mathbf{A}) \times F(\mathbf{B})) \times F(\mathbf{C}) & \xrightarrow{\cong} & F(\mathbf{A}) \times (F(\mathbf{B}) \times F(\mathbf{C})) \\
\downarrow{\scriptstyle \alpha_{\mathbf{A},\mathbf{B}}} & & \downarrow{\scriptstyle \mathrm{id} \times \alpha_{\mathbf{B},\mathbf{C}}} \\
F(\mathbf{A} \times \mathbf{B}) \times F(\mathbf{C}) & & F(\mathbf{A}) \times F(\mathbf{A} \times \mathbf{B}) \\
\downarrow{\scriptstyle \alpha_{\mathbf{A}\times\mathbf{B},\mathbf{C}}} & & \downarrow{\scriptstyle \alpha_{\mathbf{A},\mathbf{B}\times\mathbf{C}}} \\
F((\mathbf{A} \times \mathbf{B}) \times \mathbf{C}) & \xrightarrow[\cong]{} & F(\mathbf{A} \times (\mathbf{B} \times \mathbf{C}))
\end{array}$$

$$\begin{array}{ccc}
\mathbf{1} \times F(\mathbf{A}) & \xrightarrow{\epsilon \times \mathrm{id}} & F(\mathbf{1}) \times F(\mathbf{A}) \\
\downarrow{\scriptstyle \cong} & & \downarrow{\scriptstyle \alpha_{\mathbf{1},\mathbf{A}}} \\
F(\mathbf{A}) & \xleftarrow[\cong]{} & F(\mathbf{1} \times \mathbf{A})
\end{array}
\qquad
\begin{array}{ccc}
F(\mathbf{A}) \times \mathbf{1} & \xrightarrow{\mathrm{id} \times \epsilon} & F(\mathbf{A}) \times F(\mathbf{1}) \\
\downarrow & & \downarrow{\scriptstyle \alpha_{\mathbf{A},\mathbf{1}}} \\
F(\mathbf{A}) & \xleftarrow[\cong]{} & F(\mathbf{A} \times \mathbf{1})
\end{array}$$

In consequence, when F is lax monoidal, for any n-ary morphism $f : \mathbf{A}_1 \times \cdots \mathbf{A}_n \to \mathbf{B}$, we can naturally define its lifting

$$f^\dagger : F(\mathbf{A}_1) \times \cdots \times F(\mathbf{A}_n) \to F(\mathbf{B})$$

by consecutively precomposing appropriate α on $F(f)$. The coherence condition ensures that it does not matter in which specific order the α is precomposed.

See that $\flat : \mathsf{Rep} \to \mathsf{Rep}$ is lax monoidal with $\epsilon : * \mapsto *$, and $\alpha_{\mathbf{A},\mathbf{B}} : (x, y) \mapsto (x, y)$ when $x, y \neq \flat$, and $\flat$ when $x = \flat \vee y = \flat$.

For an endofunctor $F : \mathsf{C} \to \mathsf{C}$, its *tensorial strength* is a natural transformation

$$\beta_{\mathbf{A},\mathbf{B}} : \mathbf{A} \times F(\mathbf{B}) \to F(\mathbf{A} \times \mathbf{B})$$

that satisfies the coherence conditions defined by the following diagrams:

$$\begin{array}{ccccc}
(\mathbf{A} \times \mathbf{B}) \times F(\mathbf{C}) & & \xrightarrow{\beta_{\mathbf{A}\times\mathbf{B},\mathbf{C}}} & & F((\mathbf{A} \times \mathbf{B}) \times \mathbf{C}) \\
\downarrow & & & & \downarrow \\
\mathbf{A} \times \mathbf{B} \times F(\mathbf{C}) & \xrightarrow{\mathrm{id}\times\beta_{\mathbf{B},\mathbf{C}}} & \mathbf{A} \times F(\mathbf{A} \times \mathbf{C}) & \xrightarrow{\beta_{\mathbf{A},\mathbf{B}\times\mathbf{C}}} & F(\mathbf{A} \times \mathbf{B} \times \mathbf{C})
\end{array}$$

$$\begin{array}{ccc}
\mathbf{1} \times F(\mathbf{A}) & \xrightarrow{\beta_{\mathbf{1},\mathbf{A}}} & F(\mathbf{1} \times \mathbf{A}) \\
& \searrow & \downarrow \\
& & F(\mathbf{A})
\end{array}$$

In consequence, when F is equipped with a strength β, for any n-ary morphism $f : \mathbf{A}_1 \times \cdots \mathbf{A}_n \to \mathbf{B}$, we can lift a specific domain, say i'th, by precomposing the isomorphism $\mathbf{A}_1 \times \cdots \times F(\mathbf{A}_i) \times \cdots \times \mathbf{A}_n \cong \mathbf{A}_1 \times \cdots \times \mathbf{A}_n \times F(\mathbf{A}_i)$, the natural transformation β, and the isomorphism $F(\mathbf{A}_1 \times \cdots \mathbf{A}_n \times \mathbf{A}_i) \cong F(\mathbf{A}_1 \times \cdots \mathbf{A}_n)$ on $F(f)$. Let us write the lifted morphism by

$$f^{\dagger_i} : \mathbf{A}_1 \times \cdots \times F(\mathbf{A}_i) \times \cdots \times \mathbf{A}_n \to F(\mathbf{B}) \,.$$

See that in the case of $\flat : \mathsf{Rep} \to \mathsf{Rep}$, the natural transformation $\beta_{\mathbf{A},\mathbf{B}} : (x, y) \mapsto (x, y)$ when $y \neq \flat$, and $\flat$ when $y = \flat$ is a strength.

An endofunctor $F : \mathsf{C} \to \mathsf{C}$ with natural transformations $\eta : I \to F$ (unit) and $\mu : F^2 \to F$ (multiplication) is a *monad* if the coherence conditions (i) $\mu_{\mathbf{A}} \circ \eta_{F(\mathbf{A})} = \mathrm{id}_{F(\mathbf{A})} = \mu_{\mathbf{A}} \circ F(\eta_{\mathbf{A}})$ and (ii) $\mu_{\mathbf{A}} \circ \mu_{F(\mathbf{A})} = \mu_{\mathbf{A}} \circ F(\mu_{\mathbf{A}})$ hold. In other words, the following diagrams commute.

$$\begin{array}{ccc}
F(\mathbf{A}) & \xrightarrow{\eta_{F(\mathbf{A})}} & F^2(\mathbf{A}) \\
\downarrow{\scriptstyle F(\eta_{\mathbf{A}})} & \searrow & \downarrow{\scriptstyle \mu_{\mathbf{A}}} \\
F^2(vA) & \xrightarrow{\mu_{\mathbf{A}}} & F(\mathbf{A})
\end{array}
\qquad
\begin{array}{ccc}
F^3(\mathbf{A}) & \xrightarrow{\mu_{F(\mathbf{A})}} & F^2(A) \\
\downarrow{\scriptstyle F(\mu_{\mathbf{A}})} & & \downarrow{\scriptstyle \mu_{\mathbf{A}}} \\
F^2(\mathbf{A}) & \xrightarrow{\mu_{\mathbf{A}}} & F(\mathbf{A})
\end{array}$$

When (F, η, μ) is a monad, for a mapping $f : \mathbf{A} \to F(\mathbf{B})$ to a lifted codomain, we can lift only its domain $\big(\mu_{\mathbf{B}} \circ F(f)\big) : F(\mathbf{A}) \to F(\mathbf{B})$.

A functor that is lax monoidal and has a tensorial strength is called *applicative functor*. And, a monad with a tensorial strength is called strong monad. See that a strong monad automatically is an applicative functor that we can derive the α by

$$F(\mathbf{A}) \times F(\mathbf{B}) \xrightarrow{\beta_{\mathbf{A},\mathbf{B}}} F(F(\mathbf{A}) \times \mathbf{B}) \xrightarrow{F(\beta'_{\mathbf{A},\mathbf{B}})} F(F(\mathbf{A} \times \mathbf{B})) \xrightarrow{\mu_{\mathbf{A}\times\mathbf{B}}} F(F(\mathbf{A} \times \mathbf{B}))$$

where $\beta'_{\mathbf{A},\mathbf{B}} : F(\mathbf{A}) \times \mathbf{B} \to F(\mathbf{A} \times \mathbf{B})$ is defined by $\beta_{\mathbf{A},\mathbf{B}}$ and the commutativity of products. One important remark is that applicative functors are composable whereas monads are not in general.

When we have an endofunctor, it is preferred to check if it is an applicative functor, a moand, or a strong monad. Of course, the most preferred one is a strong monad.

Consider an endofunctor $F : \mathsf{Rep} \to \mathsf{Rep}$ whose mapping on morhpisms can be extended to continuously realizable functions. In other words, consider a functor $F : \mathsf{Rep} \to \mathsf{Rep}$ where there is a mapping $\zeta_{\mathbf{A},\mathbf{B}} : |\mathbf{A} \to \mathbf{B}| \to |F(\mathbf{A}) \to F(\mathbf{B})|$ such that $\zeta_{\mathbf{A},\mathbf{B}}(f) = F(f)$ for every computable $f \in \hom_{\mathsf{Rep}}(\mathbf{A}, \mathbf{B})$. Let us call an endofunctor $F : \mathsf{Rep} \to \mathsf{Rep}$ *extensible* if the mapping appears as a morphism $\zeta_{\mathbf{A},\mathbf{B}} : (\mathbf{A} \to \mathbf{B}) \to F(\mathbf{A}) \to F(\mathbf{B})$.

Lemma 2.4. An endofunctor is strong if it is extensible.

Proof. Define

$$\beta_{\mathbf{A},\mathbf{B}} \coloneqq \lambda((x, y) : \mathbf{A} \times F(\mathbf{B})).\ \zeta_{\mathbf{B},\mathbf{A}\times\mathbf{B}}\ (\lambda(z : \mathbf{B}).\ (x, z))\ y$$

and check that the coherence conditions are satisfied.

□

Example 2.5. The lazy and co-lazy lifting functors are monads with the units and multiplications:

$$\eta^{\flat}_{\mathbf{A}} : x \mapsto x \quad \mu^{\flat}_{\mathbf{A}} : x \mapsto \begin{cases} \flat_{\mathbf{A}} & \text{if } x = \flat_{\flat\mathbf{A}}, \\ x & \text{otherwise,} \end{cases}$$

and

$$\eta^{\sharp}_{\mathbf{A}} : x \mapsto x \quad \mu^{\sharp}_{\mathbf{A}} : x \mapsto \begin{cases} \sharp_{\mathbf{A}} & \text{if } x = \sharp_{\sharp\mathbf{A}}, \\ x & \text{otherwise.} \end{cases}$$

See that the mappings are computable. The desired coherence conditions can be verified easily.

They also are extensible that the definitions of $\flat : f \mapsto \flat(f)$ and $\sharp : f \mapsto \sharp(F)$ do not require f to be computable, and the procedures of obtaining realizers of $\flat(f)$ and $\sharp(f)$ are computable.

See that the desired liftings from the examples at the beginning of this section are

$$(\square_1 + \square_2 : \mathbf{R}_{\mathrm{Cauchy}} \times \mathbf{R}_{\mathrm{Cauchy}} \to \mathbf{R}_{\mathrm{Cauchy}})^{\dagger} : \flat\mathbf{R}_{\mathrm{Cauchy}} \times \flat\mathbf{R}_{\mathrm{Cauchy}} \to \flat\mathbf{R}_{\mathrm{Cauchy}},$$

$$(\mathsf{Cond}_{\mathbf{A}} : \mathbf{2} \times \mathbf{A} \times \mathbf{A} \to \mathbf{A})^{\dagger_1} : \flat\mathbf{2} \times \mathbf{A} \times \mathbf{A} \to \flat\mathbf{A},$$

and

$$(\square^{-1} \lfloor_{\flat} : \mathbf{R}_{\mathrm{Cauchy}} \to \flat\mathbf{R}_{\mathrm{Cauchy}})^{\dagger} : \flat\mathbf{R}_{\mathrm{Cauchy}} \to \flat\mathbf{R}_{\mathrm{Cauchy}}.$$

Formally speaking, a lax monoidal functor is a tuple (F, ϵ, α), a monad is a tuple (F, η, μ), an applicative functor is a tuple $(F, \epsilon, \alpha, \beta)$, and a strong monad is a tuple $(F, \eta, \mu, \alpha, \beta)$. However, we often write F to refer to the structure without writing all the components of the structure explicitly. And,

when it is needed to refer to the component, we simply write the corresponding roman alphabet. For example, when there is a monad F in the context, we write η, for example, to refer to the unit of the structure that F represents. When there are multiple structures in the context, for example, F and G, we put superscripts to distinguish to which structure the components belong. For example, we write η^F to denote the unit of F and η^G to denote the unit of G.

2.5 Real Number Computation

2.5.1 With or Without Computational Content

Not all representations are of interest. In Example 2.2, we introduced the *standard representation* of real numbers. The terminology itself suggests that the representation is the one that we are interested in amongst many different partial surjections from $\mathbb{N}^{\mathbb{N}}$ to $\mathbb{R}$.

Definition 2.7. Consider a represented set $\mathbf{A} = (A, \delta_A)$. The nc relation of $\mathbf{A}$ is a reflexive binary relation on A defined as follows:

$$x \sqsubseteq_{\mathbf{A}} y \quad :\Leftrightarrow \quad \exists \varphi \in \delta_A^{-1}(\{x\}).\, \forall n \in \mathbb{N}.\, y \in \delta_A(\bar{\varphi}_n :: \mathbb{N}^{\mathbb{N}}) \,.$$

In words, there is a name φ of x where with any finite prefix of it, we cannot determine if φ represents x or y. An element of $x \in \mathbf{A}$ is *nc* or is without any computational content if $x \sqsubseteq_{\mathbf{A}} y$ holds for any $y \in \mathbf{A}$. A represented set $\mathbf{A}$ is *nc* or is without any computational content if every elements in $|\mathbf{A}|$ are.

In contrast, a represented set $\mathbf{A}$ is *separated* if $x \sqsubseteq_{\mathbf{A}} y$ holds if any only if $x = y$.

Example 2.6.

1. The represented empty set $\mathbf{0}$ and any represented singleton, including $\mathbf{1}$, are trivially nc.
2. The represented sets $\mathbf{2}, \mathbf{N}, \mathbf{Q}$ are separated.
3. For any represented set $\mathbf{A}$, $\flat$ is an nc element in $\flat\mathbf{A}$.
4. Consider a represented set $\mathbf{R}_{\text{naive}} = (\mathbb{R}, \delta_{\text{naive}})$ where

   $$\delta_{\text{naive}}(\varphi) = x \quad :\Leftrightarrow \quad \lim_{n\to\infty} \eta_{\mathbb{Q}}(\varphi(n)) = x \,.$$

   In other words, φ is a name of a real number x when φ is an encoding of a sequence of rational numbers that *eventually* converges to x. The represented set is nc since a finite prefix of a name does not say anything about the number that the sequence converges to.
5. The represented set $\mathbf{R}_{\text{Cauchy}}$ is separated.
6. For any represented set $\mathbf{A} = (A, \delta_A)$, there is a nc representation $\mathbf{A}_{\text{nc}} = (A, \delta_{A_{nc}})$ which is defined as follows:

   $$\varphi \Vdash_{\mathbf{A}_{\text{nc}}} x \quad :\Leftrightarrow \quad \exists \varphi'.\, \varphi' \Vdash_{\mathbf{A}} x \wedge \exists L \in \mathbb{N}^*.\, \varphi = L :: 0 :: \varphi'^{>} \,.$$

   Intuitively, 0 is a token to reset naming. When φ is a δ_A-name of x, any sequences of natural numbers that ends with $0 :: \varphi'^{>}$ is a name of x in $\mathbf{A}_{\text{nc}}$. It is nc that any finite prefix of a sequence does not determine anything of what the whole sequence represents.
7. For any nc represented set, its subrepresented set is nc. And for any nc represented sets, their product representation is also nc.

8. In contrast, for any separated represented sets, their subrepresented sets and products are separated.

Lemma 2.5. Any continuously realizable function $f : |\mathbf{A}| \to |\mathbf{B}|$ preserves the nc relation; i.e., if $x \sqsubseteq_{\mathbf{A}} y$, it holds that $f(x) \sqsubseteq_{\mathbf{B}} f(y)$.

Proof. Suppose any $x, y \in \mathbf{A}$ such that $x \sqsubseteq_{\mathbf{A}} y$. There is a name φ of x and $(\varphi_n)_{n\in\mathbb{N}}$ of y such that $\varphi_n \in \bar{\varphi}_n :: \mathbb{N}^{\mathbb{N}}$ holds for all $n \in \mathbb{N}$.

Since f is continuously realizable, there is a continuous realizer τ. Consider the name $\tau(\varphi)$ of $f(x)$. Since τ is continuous, there is the modulus of continuity $m : \mathbb{N} \to \mathbb{N}$ at φ such that $\tau(\varphi_{m(n)}) \in \overline{\tau(\varphi)}_n$ for all $n \in \mathbb{N}$. Therefore, $\tau(\varphi)$ is a name of $f(x)$ where its every cylinders contains a name of $f(y)$. □

Corollary 2.1.

1. There is no non-constant continuously realizable function from a nc represented set to a separated represented set.

2. There is no non-constant continuously realizable partial function from an nc represented set to a separated represented set.

3. There is no non-constant continuously realizable partial function from $\mathbf{A}$ to $\mathbf{B}$ when $\mathbf{A}$ contains an nc element, and $\mathbf{B}$ is separated.

The above results justify our choice of the standard representation of real numbers over the naive representation. As the naive representation is nc, it does not admit any nontrivial partial approximation of the comparison test. We cannot do any effective reasoning on the order of real numbers using the naive representation.

2.5.2 Effective Representation of Real Numbers

In this section, we pay more attention to the set of real numbers and its computational structure that representations of reals provide, hoping that there is a universal structure that is less representation specific. The set of real numbers, classically, can be characterized by the constants $0, 1 \in \mathbb{R}$, the field operators $\square_1 + \square_2, \square_1 - \square_2, \square_1 \times \square_2, \square^{-1}$, the order relation $\square_1 < \square_2$, and the completion operator lim. Hence, seeing real numbers from a computation perspective, the ideal representation of reals makes the constants and operators computable.

However, we already know from Example 2.4 that the standard representation fails on making the order relation $\square_1 < \square_2$ and the completion lim computable. It gives a question why then the representation is named *standard*. We observe that the deficiency is not due to how the standard representation is defined.

Lemma 2.6.

1. ([Wei00, Theorem 4.1.16]) There is no representation of reals that makes the binary relations $\square_1 < \square_2$ and $\square_1 = \square_2$ computable

2. Any represented set of real numbers that makes lim computable is nc.

3. There is no represented real numbers that makes lim strongly computable.

Proof.

2 Consider any representation $\mathbf{R} = (\mathbb{R}, \delta)$ of real numbers. If $\lim : \mathbb{R}^{\mathbb{N}} \rightharpoonup \mathbb{R}$ is continuously realizable, due to Lemma 2.2, $\lim : |\mathsf{Seq}(\mathbf{R})| \rightharpoonup |\mathbf{R}|$ is continuously realizable.

Let τ be a continuous realizer of lim. Consider any finite sequence of natural numbers $L \in \mathbb{N}^*$ and the preimage $\tau^{-1}(L :: \mathbb{N}^{\mathbb{N}})$. If $\delta(L :: \mathbb{N}^{\mathbb{N}})$ is not empty, there is φ where $\tau(\varphi) \in L :: \mathbb{N}^{\mathbb{N}}$. Due to the continuity of τ, there is some n such that for any name φ' of a converging sequence in $\mathsf{Seq}(\mathbf{R})$ where $m(\varphi, \varphi') < 2^{-n}$, it holds that $\tau(\varphi') \in L :: \mathbb{N}^{\mathbb{N}}$.

Due to the definition of the pairing, for any two sequences $(\varphi_i)_{i\in\mathbb{N}}$ and $(\varphi'_i)_{i\in\mathbb{N}}$, the encodings $\langle(\varphi_i)_{i\in\mathbb{N}}\rangle$ and $\langle(\varphi'_i)_{i\in\mathbb{N}}\rangle$ share the first n entries when $\varphi_i = \varphi'_i$ holds for $i \leq n$.

Therefore, for any $x \in \mathbb{R}$, we can make a name of a converging sequence whose initial n entries are identical to those of φ. I.e., there is $\varphi(x)$ where $\tau(\varphi(x)) \in L :: \mathbb{N}^{\mathbb{N}}$. Hence, we can conclude $\delta(L :: \mathbb{N}^{\mathbb{N}}) = \mathbb{R}$.

3 Assume $\mathsf{lim} : |\mathbf{R}^{\mathbf{N}}| \rightharpoonup |\mathbf{R}|$ strongly and computably realizable. Then, by Lemma 2.2, there is a continuous realizer τ for the lazy extension $\mathsf{lim}\,\lfloor_\flat : \mathsf{Seq}(\mathbf{R}) \to \flat\mathbf{R}$. Consider a rapidly converging sequence $x_1, x_2, \cdots \to y$. When φ_i is a name of x_i, it holds that $\tau(\langle(\varphi_i)_{i\in\mathbb{N}}\rangle) = 0^m :: \varphi^>$ for some $m \in \mathbb{N}$ and φ which is a name of y. Due to the continuity of τ, there is some $n \in \mathbb{N}$ such that for all $(\varphi'_i)_{i\in\mathbb{N}}$ where $d(\langle(\varphi'_i)_{i\in\mathbb{N}}\rangle, \langle(\varphi_i)_{i\in\mathbb{N}}\rangle) \leq 2^{-n}$, it holds that $\tau(\langle(\varphi'_i)_{i\in\mathbb{N}}\rangle)(m) = \varphi(0) + 1 \neq 0$. However, due to the definition of $\langle.\rangle$, there is a name of $x_1, x_2, \cdots, x_n, 0, 1, 0, 1, \cdots$ whose encoding is not far from $\langle(\varphi_i)_{i\in\mathbb{N}}\rangle$ by more than 2^{-n}. On the name, τ has to print $0^{\mathbb{N}}$, but it does not.

□

Regardless of how real numbers are represented, we cannot make the order relations $\square_1 < \square_2, \square_1 = \square_2$ computable. Though there can be a representation that makes lim computable, it is an nc representation that does not admit any nontrivial partial computable function to a separated represented set. That means, in such representations, we cannot compute any partial approximation of $\square_1 < \square_2$. The observation forces us to tailor the requirement of a representation of real numbers being ideal.

Definition 2.8. A representation of reals is *effective* if the followings hold:

1. The constants $0, 1 \in \mathbb{R}$ are computable.
2. The binary operators $\square_1 + \square_2 : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, $\square_1 - \square_2 : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, and $\square_1 \times \square_2 : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ are computable.
3. The partial unary operator $\square^{-1} : \mathbb{R} \rightharpoonup \mathbb{R}$ that is not defined at 0 is strongly computable.
4. The partial binary relation $\square_1 < \square_2 : \mathbb{R} \times \mathbb{R} \rightharpoonup 2$ that is not defined at $\{(x, x) \mid x \in \mathbb{R}\}$ is strongly computable with regards to $\mathbf{2}$.
5. The partial operator $\mathsf{lim} : \mathbb{R}^{\mathbb{N}} \rightharpoonup \mathbb{R}$ that is defined only at rapidly converging sequences is weakly computable with regards to $\mathbf{N}$.

Example 2.7. The standard representation of reals is effective.

In fact, there are many different representations that are also widely used, e.g., signed digit representations, Dedekind's cut representation, regular Cauchy representation, and so on. For example, often used dyadic Cauchy represented set $\mathbf{R}_{\mathrm{dyadic}}$ is a represented set of real numbers such that

$$\varphi \Vdash_{\mathbf{R}_{\mathrm{dyadic}}} x \quad :\Leftrightarrow \quad \forall n.\, |x - \eta_{\mathbb{Z}}(\varphi(n))/2^n| \leq 2^{-n}\ .$$

Theorem 2.4 ([Her99, Theorem 3.5])**.** Any effective representations of real numbers are computably isomorphic; i.e., if δ_1 and δ_2 are effective representations of reals, there is a computable function $\tau_1, \tau_2 : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ where for any $x \in \mathbb{R}$, it holds that $\delta_2(\tau_1(\varphi)) = x$ for all $\varphi \in \delta_1^{-1}(\{x\})$ and $\delta_1(\tau_2(\varphi)) = x$ for all $\varphi \in \delta_2^{-1}(\{x\})$.

In most cases, amongst the popular representations of real numbers, it does not really matter which to use because they are all effective. That means they, including $\mathbf{R}_{\text{dyadic}}$, are isomorphic objects in Rep anyways.

2.6 Nondeterminism

Nondeterminism is essential in computable analysis. In the abstract level, a computation from a set A to another set B is nondeterministic if for a same $x \in A$, there are several values in B that the computation on x may yield. This can be identified by a nonempty set-valued function $f : A \to \mathcal{P}_\star(B)$ where $f(x)$ denotes the set of possible outputs. However, it should be clear that such a set-valued function is not what we are going to compute; i.e., we are not going to define a representation on the set $\mathcal{P}_\star(B)$. In order to make this distinction, realizing a function and realizing a nondeterministic function, clear, there is a notion of multifunction.

Definition 2.9. A *multifunction* $f : A \rightrightarrows B$ is a nonempty set-valued function $f : A \to \mathcal{P}_\star(B)$. Given representations δ_A of A and δ_B of B, the multifunction is *realized* by $F : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ if for any $x \in A$ and its realizer $\varphi \Vdash_{(A,\delta_A)} x$, it holds that $\exists y \,.\, F(\varphi) \Vdash_{(B,\delta_B)} y \wedge y \in f(x)$. Similarly to the case of ordinary functions, we say f is *continuously realized* by F if F is continuous and f is *computably realized* by F if F is computable.

Note that the definition of realizing multifunctions above specify our notion of nondeterminism. For any x, the set $f(x) \subseteq |B|$ is the set of possible results that f is expected to return regarding *nondeterminism*. Note that for a name φ of $x \in A$, the realizer F only returns one element out of the set $f(x)$. However, the same F on a different name φ' of the same input x can possibly return a name of a different element in $f(x)$. Observe that in the level of implementation, nondeterminism does not occur. It is crucial to distinguish this from the notion of nondeterminism caused by nondeterministic machines [Zie05] that is not dealt in this dissertation.

A multifunction is *partial* from A to B if it is a multifunction from a subset of A to B. See that a partial multifunction can be identified with a set-valued function which is the empty set when the input is not in its domain. We write $f :\subseteq A \rightrightarrows B$ to denote that f is a partial multifunction from A to B. As it was for partial functions, there are two different notions of realizing partial multifunctions.

Definition 2.10.

1. A partial multifunction $f :\subseteq |\mathbf{A}| \rightrightarrows |\mathbf{B}|$ is realized by $F : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ if f as a multifunction from $|\mathsf{sub}_{\mathrm{dom}(f)}(\mathbf{A})|$ to $|\mathbf{B}|$ is realized by F.
2. A partial multifunction $f :\subseteq |\mathbf{A}| \rightrightarrows |\mathbf{B}|$ is *strongly* realized by F if for any $x \in \mathbf{A}$ that is not in $\mathrm{dom}(f)$, $F(\varphi)$ diverges for any $\varphi \Vdash_{\mathbf{A}} x$; i.e., $\mathrm{dom}(F) = \delta_{\mathbf{A}}^{-1}(\mathrm{dom}(f))$.

Remark 2.3. Though the data that defines a multifunction f is a set-valued function, we refrain from writing or defining a multifunction as the set-valued function in order to avoid possible misunderstanding. When we have a set-valued function from A to nonempty subsets of B, there are two different notions

of realization: (1) realizing it as a function from A to $\mathcal{P}_\star(B)$ as in Subsection 2.3.1 and (2) realizing it as a multifunction from A to B as in Definition 2.9. We write $f : A \rightrightarrows B$ to explicitly say that we are interested in it to be realized in the second fashion.

Hence, instead of defining it as a function to sets, we often define a (partial) multifunction using the notation:

$$f : x \Mapsto \begin{cases} y_1 & \text{if } P_1 \ , \\ y_2 & \text{if } P_2 \ , \\ \vdots & \\ y_d & \text{if } P_d \ . \end{cases}$$

Here, y_i is some expression in x and P_i is some proposition in x. The notation captures nondeterminism in that when there are multiple P_i and P_j hold, the function value is y_i or y_j nondeterministically. Formally, the (partial) multifunction defined by the above notation is $f : x \mapsto \{y_i \mid P_i\}$ (that is defined at $\{x \mid f(x) \neq \emptyset\}$).

Example 2.8.

1. Given any function $f : |\mathbf{A}| \to |\mathbf{B}|$ that is realized by some $F : \mathbb{N}^\mathbb{N} \rightharpoonup \mathbb{N}^\mathbb{N}$, the multifunction $g : x \Mapsto f(x)$ is realized by F.
2. Given any multifunction $f : |\mathbf{A}| \rightrightarrows |\mathbf{B}|$ that is realized by some $F : \mathbb{N}^\mathbb{N} \rightharpoonup \mathbb{N}^\mathbb{N}$, a multifunction $g : |\mathbf{A}| \rightrightarrows |\mathbf{B}|$ such that $\forall x.\ f(x) \subseteq g(x)$ holds, is realized by F.
3. Given any multifunction $f : |\mathbf{A}| \rightrightarrows |\mathbf{B}|$ that is realized by some $F : \mathbb{N}^\mathbb{N} \rightharpoonup \mathbb{N}^\mathbb{N}$ and $g : |\mathbf{A}| \rightrightarrows |\mathbf{B}|$ that is realized by some $G : \mathbb{N}^\mathbb{N} \rightharpoonup \mathbb{N}^\mathbb{N}$, the multifunction $g \circ f$ defined by $x \mapsto \bigcup_{y \in f(x)} g(y)$ is realized by $G \circ F$.

Many essential functions are partial. For example, the order relation of real numbers can only be partially computed under the standard representation. When a partial function is strongly computable, the function gives us some more information on what the inputs are.

Example 2.9. For any natural number $n \geq 1$, the following choice operator

$$\begin{array}{rcccl} \mathsf{choice}_n & :\subseteq & |\flat\mathbf{2} \times \cdots \times \flat\mathbf{2}| & \rightrightarrows & |\mathbf{N}| \\ & := & (b_1, \cdots, b_n) & \Mapsto & \begin{cases} 1 & \text{if } b_1 = tt, \\ 2 & \text{if } b_2 = tt, \\ \vdots & \\ n & \text{if } b_n = tt. \end{cases} \end{array}$$

is strongly computable. See that the multifunction is not defined at $\{(b_1, \cdots, b_n) \mid \forall i.\ b_i \neq tt\}$. It receives finite lazy boolean values and nondeterministically pick the index of a lazy boolean, which happens to be tt. If there is no lazy boolean that is tt, it returns $\flat$.

Also, its countable version

$$\begin{array}{rcccl} \mathsf{choice}_\mathbb{N} & : & |\mathbf{N} \to \flat\mathbf{2}| & \rightrightarrows & |\mathbf{N}| \\ & := & (b_i)_{i \in \mathbb{N}} & \mapsto & \{i \mid b_i = tt\} \end{array}$$

is strongly computable. See that the countable choice function is not defined at $\{(b_i)_{i \in \mathbb{N}} \mid \forall i.\ b_i \neq tt\}$.

The nondeterministic choice is often used in practice; for example, we can approximate the sign of a real number overcoming the partiality of testing the order relation of real numbers.

Example 2.10 (Operations using $\mathbf{R}_{\mathrm{Cauchy}}$.)**.** The soft sign is a multifunction

$$\mathsf{sign} : |\mathbf{R}_{\mathrm{Cauchy}} \times \mathbf{N}| \rightrightarrows |\mathbf{2}| \coloneqq (x, n) \mapsto \begin{cases} tt & \text{if } x > -2^{-n} \\ ff & \text{if } x < 2^{-n} \end{cases}$$

that approximates the sign of a real number with a tolerance factor 2^{-n}. It is computable by precomposing the pair of $(x, n) \mapsto x{>}\lfloor_\flat\ -2^{-n}$ and $(x, n) \mapsto x{<}\lfloor_\flat\ 2^{-n}$ and postcomposing $m \mapsto \{m = 1\}$ to choose_2 .

Remark 2.4. With a similar reason to the case of continuously realizable partial functions, due to the cardinality issue, the set of continuously realizable multifunctions cannot be represented in general. In consequence, we cannot use the internal language of Rep to construct computable multifunctions. For example, one would expect a lambda term such as

$$\lambda(x : \mathbf{R}_{\mathrm{Cauchy}}).\ \mathsf{lim}(\lambda(n : \mathbf{N}).\ \mathsf{Cond}_{\mathbf{R}_{\mathrm{Cauchy}}}(\mathsf{sign}(x, n+1), x, -x)$$

to denote the absolute value function. However, since sign is not a proper morphism Rep, the above term is not permitted. This leads us to work on a more general setting where computable multifunctions appear as morphisms.

2.7 $\mathbf{Asm}(\mathbb{N}^{\mathbb{N}})$ the Category of Assemblies over $\mathbb{N}^{\mathbb{N}}$

Assembly is defined over a Partial Combinatory Algebra (PCA) where PCA provides a model of computation. However, since we are only interested in this specific model of computation in this dissertation, which is a type-2 machine over $\mathbb{N}^{\mathbb{N}}$, instead of presenting the general definition of assemblies, let us directly head to the category of assemblies over $\mathbb{N}^{\mathbb{N}}$ (Kleene's second algebra).

Definition 2.11.

- An assembly over Kleene's second algebra is a pair $\mathbf{A} \coloneqq (|\mathbf{A}|, \Vdash_{\mathbf{A}})$ of a set $|\mathbf{A}|$ and a relation $\Vdash_{\mathbf{A}} \subseteq \mathbb{N}^{\mathbb{N}} \times |\mathbf{A}|$ which is surjective in the sense that for all $x \in |\mathbf{A}|$ there is $\varphi \in \mathbb{N}^{\mathbb{N}}$ where $(x, \varphi) \in \Vdash_{\mathbf{A}}$. We write $\varphi \Vdash_{\mathbf{A}} x$ to denote $(x, \varphi) \in \Vdash_{\mathbf{A}}$ and say φ represents x or φ is a $\mathbf{A}$-name of x.

- For two assemblies $\mathbf{A}, \mathbf{B}$, a set-theoretic function $f : |\mathbf{A}| \to |\mathbf{B}|$ is said *realized* or *tracked* by $F : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ if
$$\forall x \in |\mathbf{A}|.\ \forall \varphi \in \mathbb{N}^{\mathbb{N}}.\ \varphi \Vdash_{\mathbf{A}} x \Rightarrow \tau(\varphi) \Vdash_{\mathbf{B}} f(x)$$
holds. We write $F \Vdash_{\mathbf{A} \to \mathbf{B}} f$ in the case. We say such f is continuously (computably) realized or tracked by F if F is continuous (computable). A computably realizable function is called computable

- Let us write $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ ($\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})_{\mathsf{cont}}$) for the category of assembly with computably (continuously) realizable functions as morphisms.

We drop subscripts when they is clear from contexts.

Otherwise it is explicitly mentioned, we work on $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$. Since we are not dealing with any other PCAs in this dissertation, we simply say assembly to refer to assembly over $\mathbb{N}^{\mathbb{N}}$.

Example 2.11.

1. By definition, any represented set is an assembly. And, any morphism from and to represented sets are already a morphism in Rep. In other words, Rep is a full subcategory of $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$.

2. For any set A, there is a trivial assembly ∇A defined by

$$\nabla A := (A, \mathbb{N}^{\mathbb{N}} \times A) .$$

   Any set-theoretic function $f : A \to B$ is computable as a morphism from ∇A to ∇B. In other words, the category of sets Set is a full subcategory of $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$

3. There is a forgetful functor $\mathbf{\Gamma} : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Set}$ where $\mathbf{\Gamma}(\mathbf{A})$ is the underlying set $|\mathbf{A}|$ and $\mathbf{\Gamma}(f)$ is the function f. The forgetful functor is left adjoint to ∇.

The category of assemblies $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ also satisfies the properties that we were interested in for Rep.

Remark 2.5. The category of assemblies $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ is Cartesian closed with the following properties.

1. $\mathbf{0}$ is the initial assembly and $\mathbf{1}$ is a terminal assembly.

2. For any two assemblies $\mathbf{A}, \mathbf{B}$, the assembly $\mathbf{A} \times \mathbf{B}$ on the set-theoretic Cartesian product $|\mathbf{A}| \times |\mathbf{B}|$ where

$$\langle \alpha, \beta \rangle \Vdash_{\mathbf{A}\times\mathbf{B}} (x, y) \quad :\Leftrightarrow \quad \alpha \Vdash_{\mathbf{A}} x \wedge \beta \Vdash_{\mathbf{B}} y$$

   is a category-theoretic product of $\mathbf{A}$ and $\mathbf{B}$. For any assembly $\mathbf{C}$, and morhpisms $f : \mathbf{C} \to \mathbf{A}$ and $g : \mathbf{C} \to \mathbf{B}$, let us write $f \times b : \mathbf{C} \to \mathbf{A} \times \mathbf{B}$ for the unique morphism.

3. For any two assemblies $\mathbf{A}, \mathbf{B}$, the assembly $\mathbf{A} + \mathbf{B}$ on the set-theoretic disjoint union $|\mathbf{A}| + |\mathbf{B}|$ where

$$n :: \varphi \Vdash_{\mathbf{A}+\mathbf{B}} (n, x) \quad :\Leftrightarrow \quad (n = 0 \vee \varphi \Vdash_{\mathbf{A}} x) \vee (n = 1 \vee \varphi \Vdash_{\mathbf{B}} x)$$

   is a category-theoretic coproduct of $\mathbf{A}$ and $\mathbf{B}$. For any assembly $\mathbf{C}$, and morphisms $f : \mathbf{A} \to \mathbf{C}$ and $g : \mathbf{B} \to \mathbf{C}$, let us write $f + g : (\mathbf{A} + \mathbf{B}) \to \mathbf{C}$ for the unique morphism.

4. For an assembly $\mathbf{B}$, a set A, and an injective function $\iota : A \to |\mathbf{B}|$, the subassembly induced by ι is the assembly $\mathbf{A}$ on A defined by

$$\varphi \Vdash_{\mathbf{A}} x \quad :\Leftrightarrow \quad \varphi \Vdash_{\mathbf{B}} \iota(x).$$

   See that the injective function ι is trivially computable.

5. For any two assemblies $\mathbf{A}, \mathbf{B}$, let us write $\mathcal{C}^{\downarrow}(\mathbf{A}, \mathbf{B})$ be the set of continuously realizable functions from $|\mathbf{A}|$ to $|\mathbf{B}|$. The assembly $\mathbf{B}^{\mathbf{A}}$ on the set where

$$\varphi \Vdash_{\mathbf{B}^{\mathbf{A}}} f \quad :\Leftrightarrow \quad \boldsymbol{\eta}_{\varphi} \Vdash_{\mathbf{A}\to\mathbf{B}} f$$

   is an exponential assembly where the evaluation map is the function evaluation.

In fact, $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ is a much nicer category. It is a quasitopos with $\nabla 2$ being a weak subobject classifier. However, since we are not going to make use of the structure in this dissertation, we refer [VO08] to the interested readers and stop exploring the structure of $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ further.

Similarly to Rep, we write $\mathbf{A} \to \mathbf{B}$ for $\mathbf{B}^{\mathbf{A}}$. We consider $\to$ be right-associative, and $+, \times$ be left-associative where $\times$ has the highest, and $+$ has the lowest precedence amongst the three.

2.7.1 Partial Functions in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$

In the category of represented sets Rep, we had two endofunctors $\flat, \sharp : \mathsf{Rep} \to \mathsf{Rep}$ for classifying partial functions; for two represented sets $\mathbf{A}, \mathbf{B}$, $(\mathbf{A} \to \flat\mathbf{B})$ is a represented set of continuously and strongly realizable partial functions from $\mathbf{A}$ to $\mathbf{B}$ and $(\mathbf{A} \to \sharp\mathbf{B})$ is a represented set of continuously and weakly realizable partial functions from $\mathbf{A}$ to $\mathbf{B}$.

Remark 2.6. The definitions of the endofunctors $\flat, \sharp : \mathsf{Rep} \to \mathsf{Rep}$ (from Definition 2.4 and 2.5) do not require the representations to be functions. Hence, the definitions extend to $\flat, \sharp : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$. And, the endofunctors are strong monads with the same definitions of units, multiplications, and extensions.

An advantage of working on $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ by allowing representations to be relations is that it can also deal with sets whose cardinalities exceed the cardinality of Continuum. One example is the set of continuously realizable partial functions.

Definition 2.12. For any two assemblies $\mathbf{A}$ and $\mathbf{B}$, define an assembly of continuously realizable partial functions $\mathbf{C}(\mathbf{A}, \mathbf{B}) = (\mathcal{C}(\mathbf{A}, \mathbf{B}), \Vdash_{\mathcal{C}(\mathbf{A},\mathbf{B})})$ where

$$\varphi \Vdash_{\mathbf{C}(\mathbf{A},\mathbf{B})} f \quad :\Leftrightarrow \quad \boldsymbol{\eta}_{\varphi} \Vdash_{\mathsf{sub}_{\mathrm{dom}(f)}(\mathbf{A}) \to \mathbf{B}} f.$$

See that $\varphi \in \mathbb{N}^{\mathbb{N}}$ can represent multiple partial functions.

The natural question is if $\mathbf{C}(\mathbf{A}, \mathbf{B})$ appears as an exponential object such that a computable partial function appears as a morphism where we can refer to it in the internal lambda calculus. That is, if there is an assembly $\mathbf{C}$ such that $\mathbf{B} \to \mathbf{C} \cong \mathbf{C}(\mathbf{A}, \mathbf{B})$ where $\mathbf{C}$ is related to $\mathbf{B}$ somehow.

Definition 2.13 (General Partiality)**.** Define an endofunctor $\natural : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$. The endofunctor on an assembly $\mathbf{A}$ is $\natural\mathbf{A}$ whose underlying set is $A \cup \{\natural_{\mathbf{A}}\}$ and representation relation is

$$\begin{aligned} \varphi \Vdash_{\natural\mathbf{A}} x \quad &:\Leftrightarrow \quad \varphi \text{ contains infinitely many nonzero elements} \\ &\qquad \text{and the nonzero subsequence when it is shifted by } -1 \text{ represents } x \in \mathbf{A} \\ \varphi \vdash_{\natural\mathbf{A}} \natural_{\mathbf{A}} & \end{aligned}$$

The functor on a morphism $f : \mathbf{A} \to \mathbf{B}$ is defined to be

$$\natural(f) : x \mapsto \begin{cases} f(x) & \text{if } x \neq \natural_{\mathbf{A}}, \\ \natural_{\mathbf{B}} & \text{if } x = \natural_{\mathbf{A}}. \end{cases}$$

Suppose a computable F realizes f and φ is a name of $x \in |\natural\mathbf{A}|$. Then, the procedure of searching through φ and if the entry is 0 appending it at the output tape and if the entry is $n+1$, feeding n to F computes $\natural f$.

Again, we omit the subscript from $\natural$ when it is clear from the context or is irrelevant.

Lemma 2.7. For any assemblies $\mathbf{A}$ and $\mathbf{B}$, the assembly of continuously realizable partial functions $\mathbf{C}(\mathbf{A}, \mathbf{B})$ is isomorphic to $(\natural\mathbf{B})^{\mathbf{A}}$.

Proof. Consider $F : \mathbf{C}(\mathbf{A}, \mathbf{B}) \to (\natural\mathbf{B})^{\mathbf{A}}$ and $G : (\natural\mathbf{B})^{\mathbf{A}} \to \mathbf{C}(\mathbf{A}, \mathbf{B})$ where

$$\begin{aligned} F &: f \mapsto f \restriction_{\natural} \\ G &: g \mapsto g \restriction_{\{x \mid g(x) \neq \natural\}} \end{aligned}$$

See that $F \circ G$ and $G \circ F$ are identity functions. Hence, we only need to show that the functions are computable.

For a partial function $f \in \mathcal{C}(\mathbf{A}, \mathbf{B})$, given its name φ_f and a name φ_x of $x \in A$, we need to compute a name of $F(f)(x)$. Here, we use an informal argument on type-2 machines: simulate a universal machine to compute $\boldsymbol{\eta}_{\varphi_f}(\varphi_x)$. For each computation step, append 0 in the main output tape. When $\boldsymbol{\eta}_{\varphi_f}(\varphi_x)$ appends n in its output tape, append $n+1$ in the main output tape. There are the two cases:

1. when $x \in \mathrm{dom}(f)$, the universal machine computing $\boldsymbol{\eta}_{\varphi_f}(\varphi_x)$ prints $n_1, n_2, \cdots$ which is a name of $f(x)$ where each takes some finite computation step. Hence, the main output tape is of the form $0, \cdots, 0, (n_1+1), 0, \cdots, 0, (n_2+1), 0, \cdots$. Therefore, it is a name of $f(x)$ in $\natural\mathbf{B}$.

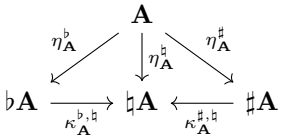
2. when $x \notin \mathrm{dom}(f)$, due to the definition of the computation, it does not fail and fill in some infinite sequence on the main output tape. Hence, it is a name of $\natural$.

For a function $g \in (\natural\mathbf{B})^{\mathbf{A}}$, given its name φ_g, and a name φ_x of $x \in \mathbf{A}$, we need to compute some $\varphi \in \mathbb{N}^{\mathbb{N}}$. The only restriction is that, due to the definition of realizing partial functions, when $x \in \mathrm{dom}(G(f))$, the computed φ has to be a name of $G(f)(x)$.

Iterate through the result of $\boldsymbol{\eta}_{\varphi_g}(\varphi_x)$. When 0 appears, ignore. When $n > 0$ appears, append $n-1$ in the output tape. When $x \in \mathrm{dom}(G(f))$, which is when $g(x) \neq \natural$, it holds that $\boldsymbol{\eta}_{\varphi_g}(\varphi_x)$ is a sequence of infinitely many nonzeros. And, its subsequence of nonzeros shifted by -1 is a name of $g(x) \in \mathbf{B}$. Hence, the computation does not fail, produces a name of $g(x)$ which is $G(f)(x)$. See that when $x \notin \mathrm{dom}(G(f))$ and when $\boldsymbol{\eta}_{\varphi_g}(\varphi_x)$ contains infinitely many zeros, this translation fails. (However, it does not matter.) □

Hence, when we want to argue the computability of a partial mapping $f : |\mathbf{A}| \rightharpoonup |\mathbf{B}|$, we can see if the extension $f \restriction_{\natural} : |\mathbf{A}| \to |\natural\mathbf{B}|$ is a morphism. The functor $\natural : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ is also a strong monad as the other partiality functors are. As the name suggests, general partiality is really general in that for a restricted partial function $f : \mathbf{A} \to \flat\mathbf{B}$ or $g : \mathbf{A} \to \sharp\mathbf{B}$, we can naturally transform them into a partial function $h : \mathbf{A} \to \natural\mathbf{B}$.

Remark 2.7. There are natural transformations $\kappa^{\flat,\natural} : \flat \to \natural$ and $\kappa^{\sharp,\natural} : \sharp \to \natural$ such that the diagram commutes.



Their definitions are

$$\kappa^{\flat,\natural}_{\mathbf{A}}(x) = \begin{cases} x & \text{if } x \neq \flat \\ \natural & \text{if } x = \flat, \end{cases} \quad \text{and} \quad \kappa^{\sharp,\natural}_{\mathbf{A}}(x) = \begin{cases} x & \text{if } x \neq \sharp \\ \natural & \text{if } x = \sharp. \end{cases}$$

2.7.2 Multifunctions in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$

One advantage of using assemblies instead of represented sets is that we can let sets whose cardinalities exceed the cardinality of Continuum be subject to computation.

The definition of realizing multifunctions in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ gets extended.

Definition 2.14. Suppose any assemblies $\mathbf{A}$ and $\mathbf{B}$ and a multifunction $f : |\mathbf{A}| \rightrightarrows |\mathbf{B}|$. A map $F : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ realizes f if for any $x \in |\mathbf{A}|$ and φ such that $\varphi \Vdash_{\mathbf{A}} x$, it holds that $F(\varphi) \Vdash_{\mathbf{B}} y$ for some $y \in f(x)$. Moreover, when f is a partial multifunction, we say F strongly realizes f if $\varphi \in \mathrm{dom}(F) \Leftrightarrow \exists x \in \mathrm{dom}(f).\ \varphi \Vdash_{\mathbf{A}} x$.

1. Let $\mathbf{M}^{\downarrow}(\mathbf{A}, \mathbf{B})$ be the assembly whose underlying set $\mathcal{M}^{\downarrow}(\mathbf{A}, \mathbf{B})$ is the set of continuously realizable multifunctions from $\mathbf{A}$ to $\mathbf{B}$ and representation relation $\Vdash$ is
$$\varphi \Vdash f \quad :\Leftrightarrow \quad \boldsymbol{\eta}_{\varphi} \text{ realizes } f\ .$$
2. Let $\mathbf{M}^{\uparrow}(\mathbf{A}, \mathbf{B})$ be the assembly whose underlying set $\mathcal{M}^{\uparrow}(\mathbf{A}, \mathbf{B})$ is the set of continuously and strongly realizable partial multifunctions from $\mathbf{A}$ to $\mathbf{B}$ and representation relation $\Vdash$ is
$$\varphi \Vdash f \quad :\Leftrightarrow \quad \boldsymbol{\eta}_{\varphi} \text{ realizes } f\ .$$

Given two assemblies, $\mathbf{A}, \mathbf{B}$, in this category, we have assemblies of continuously realizable (partial) multifunctions. Similarly to the case of general partial functions, a natural question is if they appear as exponential objects.

Definition 2.15. Define an endofunctor $\mathsf{M} : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ such that for an assembly $\mathbf{A}$, $\mathsf{M}\ \mathbf{A} \coloneqq (\mathcal{P}_{\star}(|\mathbf{A}|), \Vdash_{\mathsf{M}\ \mathbf{A}})$ where
$$\varphi \Vdash_{\mathsf{M}\ \mathbf{A}} S \quad :\Leftrightarrow \quad \exists x.\ x \in S \wedge \varphi \Vdash_{\mathbf{A}} x\ .$$
In words, φ represents a nonempty subset S of $|\mathbf{A}|$ if φ represents an element x of S with regards to the original $\mathbf{A}$.

The functor on $f : \mathbf{A} \to \mathbf{B}$, $\mathsf{M}(f) : \mathsf{M}\ \mathbf{A} \to \mathsf{M}\ \mathbf{B}$ is defined by
$$\mathsf{M}(f) : S \mapsto \bigcup_{x \in S} \{f(x)\}\ .$$
See that $\mathsf{M}(f)$ is realizable by the same realizer of f.

Lemma 2.8. The endofunctor M is a monad whose unit is $\eta_{\mathbf{A}} : x \mapsto \{x\}$ and multiplication is $\mu_{\mathbf{A}} : S \mapsto \bigcup_{T \in S} T$. See that they are realizable by $\mathrm{id} : \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$. And, it is extensible that the mapping $f \mapsto (S \mapsto \bigcup_{x \in S} \{f(x)\}$ is computable defined on all continuously realizable function. Hence, $\mathsf{M} : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ is a strong monad.

Proof. The coherence conditions are direct from the fact that the nonempty powerset functor on the category of sets is a monad. And, see that $f \mapsto (S \mapsto \bigcup_{x \in S} \{f(x)\}$ coincides with $f \mapsto F(f)$ for computable f. □

Lemma 2.9. Suppose any assemblies $\mathbf{A}$ and $\mathbf{B}$.

1. The assembly of continuously realizable mutlifunctions $\mathbf{M}^{\downarrow}(\mathbf{A}, \mathbf{B})$ is isomorphic to $\mathbf{A} \to \mathsf{M}\mathbf{B}$.
2. The assembly of continuously and strongly realizable partial mutlifunctions $\mathbf{M}^{\uparrow}(\mathbf{A}, \mathbf{B})$ is isomorphic to $\mathbf{A} \to \flat\mathsf{M}\mathbf{B}$.

Example 2.12. The multivalued choice function choice_n from Example 2.9, as a partial function from $|\flat\mathbf{2}|^n \rightharpoonup \mathsf{M}\mathbf{N}$ is stronlgy computable. That is, its lazy extension $\mathsf{choice}_n \lfloor_{\flat}$ which is
$$\begin{aligned} \mathsf{choice}_n \lfloor_{\flat} \ : \quad (\flat\mathbf{2})^n \quad &\to \quad \flat\mathsf{M}\mathbf{N} \\ : \quad (b_1, \cdots, b_n) \quad &\mapsto \quad \begin{cases} \{i \mid b_i = tt\} & \text{if } \exists i.\ b_i = tt, \\ \flat & \text{otherwise,} \end{cases} \end{aligned}$$

is computable.

See that the soft comparison operator can be defined by

$$\square_1 <_{\square_3} \square_2 \coloneqq \lambda(x\ y : \mathbf{R}_{\mathrm{Cauchy}}).\ \lambda(k : \mathbf{N}).\ (\mathsf{choice}_2 \lfloor_\flat)(x{<}\lfloor_\flat\ y + 2^{-k}, y{<}\lfloor_\flat\ x + 2^{-k}) =^{\dagger_1} 1$$

Here, $(=: \mathbf{N} \times \mathbf{N} \to \mathbf{2})^{\dagger_1} : \flat\mathsf{M}(\mathbf{N}) \times \mathbf{N} \to \flat\mathsf{M}(\mathbf{N})$ is the lifted mapping with regards to the applicative functor $\flat\mathsf{M} : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$. Recall Section 2.4 for this. See that

$$x <_k y = \begin{cases} \{tt, ff\} & \text{if } x < y + 2^{-k} \wedge y < x + 2^{-k}, \\ \{tt\} & \text{if } y \geq x + 2^{-k}, \\ \{ff\} & \text{if } x \geq y + 2^{-k}. \end{cases}$$

Similarly, the countable choice function is strongly computable. That is, its lazy extension, which is

$$\begin{array}{rcccl} \mathsf{choice}_{\mathbb{N}} \lfloor_\flat & : & (\flat\mathbf{2})^{\mathbf{N}} & \to & \flat\mathsf{M}\mathbf{N} \\ & : & (b_i)_{i\in\mathbb{N}} & \mapsto & \begin{cases} \{i \mid b_i = tt\} & \text{if } \exists i.\ b_i = tt, \\ \flat & \text{otherwise.} \end{cases} \end{array}$$

is computable.

2.7.3 Lifting Sequences

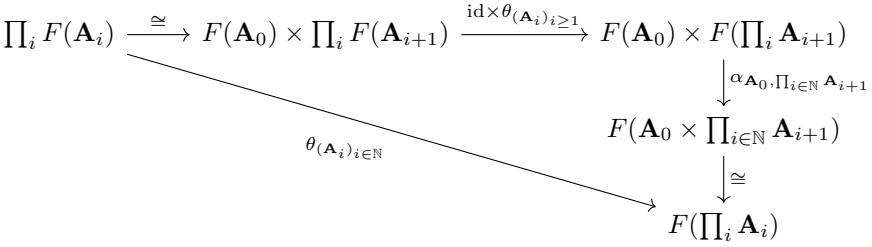
For a sequence of assemblies $(\mathbf{A}_i)_{i\in\mathbb{N}}$, see that the assembly $\prod_{i\in\mathbb{N}} \mathbf{A}_i$ over $\{(x_0, x_1, \cdots) \mid x_i \in |\mathbf{A}_i|\}$ such that

$$\langle(\varphi_i)_{i\in\mathbb{N}}\rangle \Vdash (x_i)_{i\in\mathbb{N}} \quad :\Leftrightarrow \forall i.\ \varphi_i \Vdash_{\mathbf{A}_i} x_i$$

is the countable product. See that any re-indexing and splitting to $\mathbf{A}_0 \times \prod_{i\geq 1} \mathbf{A}_i$ are isomorhpisms.

See also that $\prod_{i\in\mathbb{N}} \mathbf{A} \cong \mathbf{N} \to \mathbf{A}$. We often write $\mathbf{A}^\omega$ for $\prod_{i\in\mathbb{N}} \mathbf{A}$.

Let us call an applicative functor $(F, \epsilon, \alpha, \beta)$ *countably applicative* if for any countable product $\prod_{i\in\mathbb{N}} \mathbf{A}_i$, there is a natural transformation $\theta_{\mathbf{A}_i} : \prod_{i\in\mathbb{N}} F(\mathbf{A}_i) \to F(\prod_{i\in\mathbb{N}} \mathbf{A}_i)$ with the coherence condition:



Amongst the applicative functors we have used in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$, $\sharp$, $\natural$, and M are countably applicative with

$$\theta^{\sharp}_{(\mathbf{A}_i)_{i\in\mathbb{N}}}(x_i)_{i\in\mathbb{N}} = \begin{cases} \sharp & \text{if } \exists i.\ x_i = \sharp, \\ (x_0, x_1, \cdots) & \text{otherwise,} \end{cases}$$

$$\theta^{\natural}_{(\mathbf{A}_i)_{i\in\mathbb{N}}}(x_i)_{i\in\mathbb{N}} = \begin{cases} \natural & \text{if } \exists i.\ x_i = \natural, \\ (x_0, x_1, \cdots) & \text{otherwise,} \end{cases}$$

and

$$\theta^{\mathsf{M}}_{(\mathbf{A}_i)_{i\in\mathbb{N}}}(S_i)_{i\in\mathbb{N}} = \bigcup_{x_i\in S_i} \{(x_0, x_1, \cdots)\}.$$

However, $\flat$ is not countably applicative that given an infinite sequence of $\flat$ lifted values, we cannot semi-decide if there is $\flat$ in the sequence or not. Hence, it is evident that strong monads can fail to be strongly applicative in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$.

Luckily, countably applicative functors are composable.

Lemma 2.10. Suppose $(F, \epsilon^F, \alpha^F, \beta^F, \theta^F)$ and $(G, \epsilon^G, \alpha^G, \beta^G, \theta^G)$ are countably applicative functors. Then, the composed applicative functor FG is countably applicative with

$$\theta^{FG} = F(\theta^G) \circ \theta^F$$

Proof. For any sequence $(\mathbf{A}_i)_{i\in\mathbb{N}}$, we need to show that the diagram commutes (with omitting isomorphisms):

$$\begin{array}{ccccc}
FG(\mathbf{A}_0) \times FG(\mathbf{A}_1) \times \cdots & \xrightarrow{\mathrm{id}\times\theta^F} & FG(\mathbf{A}_0) \times F(G(\mathbf{A}_1) \times \cdots) & \xrightarrow{\mathrm{id}\times F(\theta^G)} & FG(\mathbf{A}_0) \times FG(\mathbf{A}_1 \times \cdots) \\
\downarrow{\scriptstyle \theta^F} & & & & \downarrow{\scriptstyle \alpha^F} \\
F(G(\mathbf{A}_0) \times G(\mathbf{A}_1) \times \cdots) & & & & F(G(\mathbf{A}_0) \times G(\mathbf{A}_1 \times \cdots)) \\
\downarrow{\scriptstyle F(\theta^G)} & & & & \downarrow{\scriptstyle F(\alpha^G)} \\
FG(\mathbf{A}_0 \times \mathbf{A}_1 \times \cdots) & = & = & = & FG(\mathbf{A}_0 \times \mathbf{A}_1 \times \cdots)
\end{array}$$

By the condition of θ^F, the diagram commutes:

$$\begin{array}{ccc}
FG(\mathbf{A}_0) \times FG(\mathbf{A}_1) \times \cdots & \xrightarrow{\mathrm{id}\times\theta^F} & FG(\mathbf{A}_0) \times F(G(\mathbf{A}_1) \times \cdots) \\
\downarrow{\scriptstyle \theta^F} & & \downarrow{\scriptstyle \alpha^F} \\
F(G(\mathbf{A}_0) \times G(\mathbf{A}_1) \times \cdots) & = & F(G(\mathbf{A}_0) \times G(\mathbf{A}_1) \times \cdots)
\end{array}$$

Hence, the desired diagram reduces to

$$\begin{array}{ccc}
FG(\mathbf{A}_0) \times F(G(\mathbf{A}_1) \times \cdots) & \xrightarrow{\mathrm{id}\times F(\theta^G)} & FG(\mathbf{A}_0) \times FG(\mathbf{A}_1 \times \cdots) \\
\downarrow{\scriptstyle \alpha^F} & & \downarrow{\scriptstyle \alpha^F} \\
F(G(\mathbf{A}_0) \times G(\mathbf{A}_1) \times \cdots) & & F(G(\mathbf{A}_0) \times G(\mathbf{A}_1 \times \cdots)) \\
\downarrow{\scriptstyle F(\theta^G)} & & \downarrow{\scriptstyle F(\alpha^G)} \\
FG(\mathbf{A}_0 \times \mathbf{A}_1 \times \cdots) & = & FG(\mathbf{A}_0 \times \mathbf{A}_1 \times \cdots)
\end{array}$$

As α^F is a natural transformation, the diagram commutes:

$$\begin{array}{ccc}
FG(\mathbf{A}_0) \times F(G(\mathbf{A}_1) \times \cdots) & \xrightarrow{\mathrm{id}\times F(\theta^G)} & FG(\mathbf{A}_0) \times FG(\mathbf{A}_1 \times \cdots) \\
\downarrow{\scriptstyle \alpha^F} & & \downarrow{\scriptstyle \alpha^F} \\
F(G(\mathbf{A}_0) \times G(\mathbf{A}_1) \times \cdots) & \xrightarrow{F(\mathrm{id}\times\theta^G)} & F(G(\mathbf{A}_0) \times G(\mathbf{A}_1 \times \cdots))
\end{array}$$

Therefore, the diagram again reduces to

$$\begin{array}{ccc}
F(G(\mathbf{A}_0) \times G(\mathbf{A}_1) \times \cdots) & \xrightarrow{F(\mathrm{id}\times\theta^G)} & F(G(\mathbf{A}_0) \times G(\mathbf{A}_1 \times \cdots)) \\
\downarrow{\scriptstyle F(\theta^G)} & & \downarrow{\scriptstyle F(\alpha^G)} \\
FG(\mathbf{A}_0 \times \mathbf{A}_1 \times \cdots) & = & FG(\mathbf{A}_0 \times \mathbf{A}_1 \times \cdots)
\end{array}$$

which is the condition for θ^G on F. □

In addition to other liftings, when an applicative functor F is countably applicative, for a morphism $f : \prod_{i\in\mathbb{N}} \mathbf{A}_i \to \mathbf{B}$, let us define its lifting

$$f^\dagger : \prod_{i\in\mathbb{N}} F(\mathbf{A}_i) \to F(\mathbf{B})$$

by precomposing θ^F on $F(f)$. Morevoer, when F is a countably applicative monad, for a morphism $f : \prod_{i\in\mathbb{N}} \mathbf{A}_i \to F(\mathbf{B})$, let us define its lifting

$$f^\dagger : \prod_{i\in\mathbb{N}} F(\mathbf{A}_i) \to F(\mathbf{B})$$

by precomposing θ^F and postcomposing μ^F on $F(f)$.

Let us see some examples. The most (the only to be honest) use case in the dissertation of the liftings is when the countable product is of the form $\mathbf{A}^\omega$. As $\mathbf{A}^\omega$ is isomorphic to $\mathbf{N} \to \mathbf{A}$, let us assume that the isomorphism is taken implicitly and work on $\mathbf{N} \to \mathbf{A}$ instead. Also, let us abbreviate $\theta_{\mathbf{A}}$ for $\theta_{(\mathbf{A})_{i\in\mathbb{N}}}$ in the case.

First, see that for a morhpism $f : (\mathbf{N} \to \mathbf{A}) \to \mathsf{M}\mathbf{B}$, a multifunction from sequences, its lifting $f^\dagger : (\mathbf{N} \to \mathsf{M}\mathbf{A}) \to \mathsf{M}\mathbf{B}$ happens to be

$$g \mapsto \bigcup_{x_n \in g(n)} \{f((x_i)_{i\in\mathbb{N}})\}.$$

That is, it unions over all applications of f on each section of g.

The first use case is the countable choice function

$$\mathsf{choice}_{\mathbb{N}} \lfloor_\flat : (\mathbf{N} \to \flat\mathbf{2}) \to \flat\mathbf{N}.$$

We can think of the situation where we want to pass a multivalued test sequence as its input. In this case, we can simply lift it with regards to M:

$$(\mathsf{choice}_{\mathbb{N}} \lfloor_\flat)^\dagger : (\mathbf{N} \to \mathsf{M}\flat\mathbf{2}) \to \mathsf{M}\flat\mathbf{N}\ .$$

See that the definition is

$$(\mathsf{choice}_{\mathbb{N}} \lfloor_\flat)^\dagger((S_i)_{i\in\mathbb{N}}) = \bigcup_{b_i \in S_i} \mathsf{choice}((b_i)_{i\in\mathbb{N}}).$$

It goes through all combinations of $b_i \in S_i$ and return the indices i such that $tt \in S_i$.

The second use case is the partial limit operation

$$\mathsf{lim} \lfloor_\sharp : (\mathbf{N} \to \mathbf{R}_{\mathrm{Cauchy}}) \to \sharp\mathbf{R}_{\mathrm{Cauchy}}.$$

When, we lift it, $(\mathsf{lim} \lfloor_\sharp)^\dagger : (\mathbf{N} \to \mathsf{M}\mathbf{R}_{\mathrm{Cauchy}}) \to \mathsf{M}\sharp\mathbf{R}_{\mathrm{Cauchy}}$, it is identified as follows:

$$x \in (\mathsf{lim} \lfloor_\sharp)^\dagger((S_i)_{i\in\mathbb{N}}) \Leftrightarrow \exists x_i \in S_i.\ |x - x_i| \le 2^{-i} \quad \text{and} \quad \sharp \in (\mathsf{lim} \lfloor_\sharp)^\dagger((S_i)_{i\in\mathbb{N}}) \Leftrightarrow \exists x_i \in S_i.\ \nexists x.\ |x - x_i| \le 2^{-i}\ .$$

See that when $\sharp$ is not in the result, the result has to be a singleton. Otherwise, if $x, y \in (\mathsf{lim} \lfloor_\sharp)^\dagger((S_i)_{i\in\mathbb{N}})$ such that $x \neq y$, it must be that $\sharp \in (\mathsf{lim} \lfloor_\sharp)^\dagger((S_i)_{i\in\mathbb{N}})$. Hence, there are the three cases (1) the result is $\{\sharp\}$, (2) the result is a single real number $\{x\}$, and (3) the result is multiple real numbers with $\sharp$ in it $\{\sharp, x, y \cdots\}$. This is useful when we do multivalued computation, and it is needed to compute the limit of multivalued real numbers.

Also, consider a case where we feed partial real numbers in limit. Lifting the limit function with regards to $\natural$ happens to be

$$(\mathsf{lim}\,\lfloor_\sharp)^\dagger((x_i)_{i\in\mathbb{N}}) = \begin{cases} \lim(x_i)_{i\in\mathbb{N}} & \text{if } \exists x.\ \forall i.\ |x - x_i| \leq 2^{-i}, \\ \natural & \text{if } \exists i.\ x_i = \natural, \\ \sharp & \text{otherwise.} \end{cases}$$

Of course, we can lift lim with regards to $\natural\mathsf{M}$, as $\natural\mathsf{M}$ also is a countably applicative functor, considering the situation where we want to feed in multivalued partial real numbers.

However, it is not possible to directly feed in lazy real numbers as $\flat$ is not countably applicative. Hence, when we want to feed in lazy real numbers $x \in \flat\mathbf{R}$, we need to convert it to a generally partial real number $x \in \natural\mathbf{R}$ first.

Example 2.13. Finally! we are ready to write a term that computes the absolute value function in the internal language. First, define the soft-sign function

$$\mathsf{Sign}_{\square_2}(\square_1) \coloneqq \lambda(x : \mathbf{R}_{\mathrm{Cauchy}}).\ \lambda(n : \mathbf{N}).\ (\mathsf{choice}_2\,\lfloor_\flat)(x > -2^{-n}, 2^{-n} > x) =^{\dagger_1} 1$$

as a morphism from $\mathbf{R}_{\mathrm{Cauchy}}$ to $\flat\mathsf{M}(\mathbf{2})$. Then, see that

$$\lambda(n : \mathbf{N}).\ \mathsf{Cond}^{\dagger_1}_{\mathbf{R}_{\mathrm{Cauchy}}}\big(\mathsf{Sign}(x, n+1), x, -x\big)\Big)$$

assuming $x : \mathbf{R}_{\mathrm{Cauchy}}$ is a morphism from $\mathbf{N}$ to $\flat\mathsf{M}(\mathbf{R}_{\mathrm{Cauchy}})$. Hence, postcomposing $\kappa^{\flat,\natural}_{\mathbf{R}_{\mathrm{Cauchy}}}$ yields a morhpism from $\mathbf{N}$ to $\natural\mathsf{M}(\mathbf{R}_{\mathrm{Cauchy}})$.

$$\kappa^{\flat,\natural}_{\mathbf{R}_{\mathrm{Cauchy}}} \circ \mathsf{Cond}^{\dagger_1}_{\mathbf{R}_{\mathrm{Cauchy}}}\big(\mathsf{Sign}(x, n+1), x, -x\big)$$

Therefore, feeding it to lim lifted with regards to $\natural\mathsf{M}$ yields

$$\mathsf{abs} \coloneqq \lambda(x : \mathbf{R}_{\mathrm{Cauchy}}).\ (\mathsf{lim}\,\lfloor_\sharp)^\dagger\Big(\lambda(n : \mathbf{N}).\ \kappa^{\flat,\natural}_{\mathbf{R}_{\mathrm{Cauchy}}} \circ \mathsf{Cond}^{\dagger_1}_{\mathbf{R}_{\mathrm{Cauchy}}}\big(\mathsf{Sign}(x, n+1), x, -x\big)\Big)$$

which is a morphism from $\mathbf{R}_{\mathrm{Cauchy}}$ to $\natural\mathsf{M}(\sharp\mathbf{R}_{\mathrm{Cauchy}})$ that computes the absolute value of the input.

See that $\mathsf{abs} : \mathbf{R}_{\mathrm{Cauchy}} \to \natural\mathsf{M}(\sharp\mathbf{R}_{\mathrm{Cauchy}})$ is a little disappointing that we know the absolute value function is not a (strict) multifunction or a partial function. Of course, we can prove it by reasoning on the definition of abs that for any x, $\mathsf{abs}(x)$ is $\{|x|\}$. And, this should not be hard as the definition is simple enough. However, the point is, to make sure abs is indeed what we wanted to have, we need an extra procedure. This is what leads to the main work of this dissertation, formalizing the procedure using the framework of imperative programming.

2.8 Further Remarks on Multifunctions

It is all about classifying functions. By studying computability theory, we classify functions that are computable and that are not. By studying algebra, we classify functions that are homomorphic and that are not. By studying topology, we classify functions that are continuous and that are not. The question is if we can make the notion of computability intrinsic, the set of computable functions becomes derivable from a natural structure of the sets. The main theorem in computable analysis [Wei00, Theorem 3.2.11] states it is somewhat possible in the sense that there are representations on topological spaces that make the characterizations, (i) continuously realizable and (ii) continuous, coincide.

Definition 2.16. A representation δ of a topological space X is *continuous* if it is a continuous mapping from the domain equipped with the subspace topology of the standard topology on $\mathbb{N}^{\mathbb{N}}$ to X. A continuous representation δ of a topological space X is (computably) *admissible* if for any continuous representation γ of X, the identity function $\mathrm{id} : (X, \delta) \to (X, \gamma)$ is continuously (computably) realizable.

Remark 2.8. The standard representation of reals is admissible and computably admissible.

Theorem 2.5 (The Main Theorem [Wei00])**.** Suppose any two topological spaces with representations (X, δ_X) and (Y, δ_Y). If the representations are admissible, for any function $f : (X, \delta_X) \to (Y, \delta_Y)$, the function is continuous if and only if it is continuously realizable.

Thanks to the main theorem, we do not need to talk about representations all the time. The computational structure of a set becomes intrinsic in the sense that, for an example of real numbers, when we want to mention continuously realizable functions, we can refer to the property by mentioning continuous functions. Consequently, the represented set $\mathbf{R} \to \mathbf{R}$ for any effective represented set $\mathbf{R}$, is the represented set of continuous real functions.

Once we are interested in multifunctions, a natural question to ask is if there is a topology on $\mathcal{P}_\star(Y)$ that makes the statement holds: a function $f : \mathbf{X} \to \mathsf{M}(\mathbf{Y})$ is continuously realizable if and only if it is continuous.

Lemma 2.11. Let X be a nonempty set and $\mathcal{U} \subsetneq \mathcal{P}_\star(X)$. Then there exists a pair of nonempty subsets $S, T \subseteq X$ such that $S \in \mathcal{U}$, $T \notin \mathcal{U}$, and $S \cap T \neq \emptyset$.

Proof. Suppose not. Since $\mathcal{U} \neq \emptyset$, we can pick $S \in \mathcal{U}$. Since $X \cap S \neq \emptyset$, we have $X \in \mathcal{U}$. Therefore for every nonempty subset $Y \subseteq X$, we have $Y \in \mathcal{U}$. We conclude $\mathcal{U} = \mathcal{P}_\star(X)$, a contradiction. □

Theorem 2.6. Let $(A, \Vdash_A)$ and $(B, \Vdash_B)$ be nonempty represented sets. If every continuously realizable $f : A \to \mathcal{P}_\star(B)$ is continuous, then either A has the discrete topology or $\mathcal{P}_\star(B)$ has the trivial topology.

Proof. Suppose that $\mathcal{P}_\star(B)$ has a nontrivial topology, so that there exists open $\mathcal{U}$ satisfying $\emptyset \neq \mathcal{U} \subsetneq \mathcal{P}_\star(B)$. By applying Lemma 2.11, we pick S and T such that $S \in \mathcal{U}$, $T \notin \mathcal{U}$, and $S \cap T \neq \emptyset$. Fix $a_0 \in A$ and consider the total multifunction $f : A \rightrightarrows B$

$$f(x) := \begin{cases} S & (x = a_0) \\ T & (x \neq a_0) \end{cases} .$$

f is continuously realizable by a constant realizer mapping everything to a name of $b_0 \in S \cap T$. Therefore f is continuous by one of the assumptions. By continuity of f,

$$f^{-1}[\mathcal{U}] = \{a_0\}$$

is open, which is a one-point set. Since choice of a_0 was arbitrary, we conclude that A has the discrete topology. □

Corollary 2.2. When A and B are of the cardinality of continuum, for any representations of A and B, there are no topologies on A and $\mathcal{P}_\star(B)$ that make the statement hold:

$$f : A \rightrightarrows B \text{ is continuous if and only if } f \text{ is continuously realizable.}$$

Proof. Assume that the statement holds. Then, by Theorem 2.6, every multifunctions are continuous. By the assumption, every multifunctions are continuously realizable. For any function $f : A \to B$, there is a continuous realizer τ that realizes $\bar{f} : x \mapsto \{f(x)\}$. Note that τ realizes f as well. Hence, any function from A to B becomes continuously realizable. However, since the cardinality of the set of continuous functions from A to B strictly exceeds the cardinality of the set of continuous functions from $\mathbb{N}^{\mathbb{N}}$ to $\mathbb{N}^{\mathbb{N}}$, it is a contradiction. □

Chapter 3. ERC: Simple Imperative Language with Real Numbers

In this chapter, we devise the simple imperative language ERC (Exact Real Computation). The language provides the primitive data type R for real numbers and primitive operators for its field arithmetic: $+, -, \times$, and $\square^{-1}$. Their semantics is defined to be exactly the field arithmetic of real numbers.

Of course, we want the language to be implementable. And, we know from Chapter 2 how computation over real numbers needs to be performed. That is, the language has to be based on type-2 computation. But what does it mean for a language to be based on type-2 computation?

It is a typed imperative language. That is, there is a variable x of type R. It is a store that holds a datum that represents a value in $\mathbb{R}$. It means that there is a representation δ of real numbers underlies such that in the implementation level, x is storing a datum $\varphi \in \mathbb{N}^{\mathbb{N}}$, meanwhile, in the abstract level, we see the datum as $\delta(\varphi) \in \mathbb{R}$.

In this framework, it is clear what is going on with the following instruction where x, y, z are variables of type R.

$$z := x + y$$

Suppose x and y are storing $\varphi_x \in \mathbb{N}^{\mathbb{N}}$ and $\varphi_y \in \mathbb{N}^{\mathbb{N}}$ in the implementation level. Then, when we execute the above instruction, it runs a realizer of the real number addition on φ_x and φ_y, and assign the result to the store z. In the abstract level, we see that the value $\delta(\varphi_x) + \delta(\varphi_y)$ is assigned at z.

It is important that though we want an implementable language, implementation is not a part of the definition of the language. For example, we want the user of the language to see x in the above example as a variable that is storing a real number in $\mathbb{R}$ not an infinite sequence in $\mathbb{N}^{\mathbb{N}}$. When the user of the language writes the above program, we want it to be seen as the real number addition $x + y$, not the realizer of the addition.

We define the semantics of the language without any implementation-specific details. For example, the semantics of a variable of type R is a store that contains real numbers, and the semantics of R operations are the real number operations. Hence, users can write a program without considering anything about infinite sequences, and they can reason on their programs by their mathematical knowledge of real numbers. However, at the end of the day, due to the property being implementable, realizers of the programs which computers can simulate will be obtained.

Obviously, the language should provide an operator for real number comparisons. However, having the ordinary total real number comparison $\square_1 < \square_2 : \mathbb{R}^2 \to \mathbb{2}$ is not feasible as it won't be implementable anyway (recall Example 2.6).

Recall from Example 2.4, that instead, the partial comparison $\square_1 < \square_2 : \mathbb{R}^2 \rightharpoonup \mathbb{2}$ which is not defined at $\{(x, x) \mid x \in \mathbb{R}\}$ is feasible. That is, we can define $z := x < y$ to correctly assign tt at z when $x < y$, ff at z when $y < x$, but be unspecified when $x = y$. However, we can do more than this since the partial function $<$ is *strongly* computable. That is, its lazy extension $\square_1 <\!\lfloor_\flat\, \square_2 : |\mathbf{R}^2| \to |\flat\mathbf{2}|$ is computable.

We equip the language with the lazy extension and the lazy lifted Boolean. Instead of writing $|\flat\mathbf{2}|$, in order to simplify the syntax presentation of our language, let us write $\mathbb{K} = \{tt, ff, uk\}$ where uk corresponds to $\flat \in |\flat\mathbf{2}|$. The symbol $\mathbb{K}$ stands for **K**leene logic where uk stands for the third truth-value

in Kleene logic "unknown". Similarly, we write $\lesssim$ instead of $<\!\!\lfloor_\flat$:

$$x \lesssim y = \begin{cases} tt & \text{if } x < y, \\ ff & \text{if } y < x, \\ uk & \text{otherwise.} \end{cases}$$

The added element uk in $\mathbb{K}$ denotes nontermination in a safe way. For example, when b is a variable of type K, the data type for $\mathbb{K}$, the statement $b \coloneqq x \lesssim y$ safely assigns uk at b when $x = y$ without diverging. However, when it comes a moment where we need to decide if it is tt or ff, it makes the program diverge. For example, the instructions

$$\textbf{if } b \texttt{ then } c_1 \textbf{ else } c_2 \quad \text{and} \quad \textbf{while } b \textbf{ do } c$$

diverge when $b = uk$.

Nondeterminism, which is essential in exact real number computation [Luc77], is provided by countably many constructs. There is a construct $\texttt{choose}_n$ for each natural number $n \in \mathbb{N}$. This iRRAM-like [Mül00] construct $\texttt{choose}_n(b_1, \cdots, b_n)$ receives n terms of type K and evaluates to the index of a term which evaluates to tt. When there are multiple arguments that evaluate to tt, it returns any of those indices *nondeterministically*. For example, $\texttt{choose}_2(\texttt{true}, \texttt{true})$, where $\texttt{true}$ is a programming constant for tt, evaluates to either 1 or 2 nondeterministically. Even when one of its arguments evaluates to uk, as long as there is an argument that evaluates to tt, the whole expression does not diverge. See Example 2.9.

Using this construct, we can construct a term for the soft comparison test with a tolerance factor ϵ as follows:

$$\texttt{choose}_2(x \lesssim y + \epsilon, y \lesssim x + \epsilon) \mathrel{\hat{=}} 1 \ .$$

Here, $\square_1 \mathrel{\hat{=}} \square_2 : \mathbb{Z} \times \mathbb{Z} \to \mathbb{K}$ is the integer equality test postcomposed by the subset inclusion. Similarly, let us define $\square_1 \mathrel{\hat{\lesssim}} \square_2 : \mathbb{Z} \times \mathbb{Z} \to \mathbb{K}$. The above term evaluates to tt when $x < y + \epsilon$ and to ff when $y < x + \epsilon$. When both inequalities hold, one of the two gets returned nondeterministically.

Constructing real numbers via the limits of converging sequences is only done implicitly. Suppose we have an explicit limit operator $\texttt{lim}$ for the feature. Then, the operator has to receive a function, an infinite sequence, or something that expresses an infinite sequence of approximations. In order to make our language as simple as possible, we take a similar approach to [TZ99, TZ04, TZ15]. When we have a program that computes a real number from a natural number, we add a layer of interpretation in that we regard the program as the limit of the sequence generated by altering the input natural number.

The extended language being nondeterministic, we use Plotkin powerdomain [Plo76] to interpret our denotational semantics. Given a term, the denotation of it is defined to be the set of values that each nondeterministic branch in the evaluation results. For example, the denotation of $\texttt{choose}_2(\texttt{true}, \texttt{true})$ on any state is the set $\{1, 2\}$.

Other than the enriched term language, commands are identical to the simple imperative language. Hence, we use precondition-postcondition-style program specification in order to describe the property of a program. We choose the first-order logic of the structures of Presburger arithmetic, ordered field of real numbers, and Kleene logic connected by the *accuracy embedding* $2^{\square} : \mathbb{Z} \ni p \mapsto 2^p \in \mathbb{R}$ to be the logical language expressing preconditions and postconditions. We show that the logical language is expressive enough to express the denotations of our term language. In other words, there is a recursive translation from the set of terms to the set of formulae such that the translated formula of a term, with regards to the standard interpretation, defines the term's denotation. We show that the theory of the logical

language is complete and decidable; i.e., any sentence in the language can be automatically proved or disproved. (For this, the two structures being connected by the accuracy embedding is crucial.)

We devise Hoare-style proof rules for proving correct specification. We prove that the proof rules, as a formal system, is sound with regards to the denotational semantics. However, as the inevitable side-effect of the logical language being complete, the formal system is not complete in the sense of [Coo78]. However, by providing an example proof of the correctness of a root-finding algorithm in the next chapter, we demonstrate the practical usefulness of our design.

In Section 3.1, we introduce ERC as a programming language with some demonstrative examples. In the section, we deliver the intended meaning of each construct informally. In Section 3.2, we define the formal syntax and the type system of ERC. In Section 3.3, we define domain-theoretic denotational semantics and prove that our language is Turing-complete. In Section 3.4, we devise a specification language and prove that the language is complete and decidable. After seeing that the language is expressive for the term language of ERC, we devise Hoare-style proof rules. We prove that they, as a formal system, are sound regarding the denotational semantics. The Computability and implementabiltiy issues are dealt with in the next chapter.

3.1 Overview of ERC with Example Programs

In this section, we overview ERC with some example programs. The intended meaning of each construct in ERC is explained.

A program in ERC is in the following form:

$$\begin{array}{l}\textbf{function}\ (x_1 : \tau_1, \cdots, x_d : \tau_d)\\ \quad c\\ \textbf{return}\ t\end{array}$$

Here, x_i is an input variable of type τ_i, c is a command, and t is a term that to be returned.

$$\begin{array}{ll}\mathsf{partial\_abs} := & \textbf{function}\ (x : \mathsf{R})\\ & \quad \textbf{var}\ y : \mathsf{R} := 0;\\ & \quad \textbf{if}\ x \gtrsim 0\ \textbf{then}\ y := x\ \textbf{else}\ y := -x\\ & \textbf{return}\ y\end{array}$$

The program, which is named partial_abs receives a real number x. It first creates a new variable y declaring its type to be R, and it initializes y to be 0. Commands get sequentially composed by ';'. Hence, the program says, after creating the variable, it tests if $x > 0$. If $x > 0$ holds, the term $x \gtrsim 0$ evaluates to tt and the first branch is taken. And, it assigns the value of x at y. If $x < 0$ holds, the term $x \gtrsim 0$ evaluates to ff and the second branch is taken. In this case, it assigns the value of $-x$ at y. If $x = 0$, since $x \gtrsim 0$ is in the condition, the program diverges. Hence, the program partial_abs computes the absolute value of its input exactly but partially in that when the input is identical to zero, it never terminates.

The use of nondeterminism enables us to make a total program for computing the absolute values:

$$
\begin{aligned}
\mathsf{abs} \coloneqq\ & \textbf{function}\ (p : \mathsf{Z}, x : \mathsf{R}) \\
& \quad \textbf{var}\ y : \mathsf{R} \coloneqq 0; \\
& \quad \textbf{if}\ \mathtt{choose}_2(x \gtrsim -2^{p-1}, x \lesssim 2^{p-1}) \mathrel{\hat{=}} 1. \\
& \quad\ \textbf{then}\ y \coloneqq x \\
& \quad\ \textbf{else}\ y \coloneqq -x \\
& \textbf{return}\ y
\end{aligned}
$$

It evaluates $\mathtt{choose}_2(x \gtrsim -2^{p-1}, x \lesssim 2^{p-1})$. Given an integer p, the term 2^p, which is for the accuracy embedding, evaluates to 2^p of type R. And, `choose` nondeterministically returns the index of an argument which evaluates to *tt*. See that for any x and p at least one of the arguments $x \gtrsim -2^{p-1}$ and $x \lesssim 2^{p-1}$ evaluates to *tt*. Hence, the `choose` term evaluates to either 1 or 2 and always terminates. When 1 is returned, it means that $x > -2^{p-1}$; and, when 2 is returned, it means that $x < 2^{p-1}$. Hence, whichever nondeterministic branch is taken, at the end of the program, y stores a 2^p-approximation to $|x|$. When we have a program whose return type is R and the first argument is of type Z, we regard the integer input as the precision parameter. And, we can interpret the program to computing the limit value of the sequence generated by sending the first argument to $-\infty$. Hence, in this case, we say the program abs computes the absolute function.

Consider the program for computing the multivalued rounding:

$$
\begin{aligned}
\mathsf{round} :\equiv\ & \textbf{function}\ (x : \mathsf{R}) \\
& \quad \textbf{var}\ k : \mathsf{Z} \coloneqq 0; \\
& \quad \textbf{while}\ \mathtt{choose}_2(x \lesssim 1, x \gtrsim 1/2) \mathrel{\hat{=}} 2\ \textbf{do} \\
& \quad\quad k \coloneqq k + 1; \\
& \quad\quad x \coloneqq x - 1 \\
& \quad \textbf{while}\ \mathtt{choose}_2(x \gtrsim -1, x \lesssim -1/2) \mathrel{\hat{=}} 2\ \textbf{do} \\
& \quad\quad k \coloneqq k - 1; \\
& \quad\quad x \coloneqq x + 1; \\
& \textbf{return}\ k
\end{aligned}
$$

The program round returns a multivaleud integer k where $x - 1 < k < x + 1$ holds.

3.2 Formal Syntax and Typing

In this section, we construct ERC as a formal programming language.

3.2.1 Formal Syntax

Programs in ERC are constructed using the three layers: terms, commands, and programs. Terms in ERC represent mathematical values, commands in ERC represent computational instructions on how to alter computer states, and programs in ERC represent functions with inputs and an output.

Data Types in ERC

A term in ERC represents mathematical values that to be expressed in ERC. Data types are the domains of the values. ERC as a formal programming language provides the three data types: Z for

integers, R for real numbers, and K for Kleeneans. We often write τ and its variants to denote an arbitrary data type.

Terms in ERC

The term language that ERC provides is Presburger arithmetic, Real Closed Field, and Kleene Logic. As usual, we assume that there are unlimited supplies of variables. The terms in ERC are defined inductively as follow:

t	$::= x$	variable
	$\mid$ `true` $\mid$ `false` $\mid$ `undef`	K constant
	$\mid k_{\mathsf{Z}}$	integer constant $k \in \mathbb{Z}$
	$\mid k_{\mathsf{R}}$	real number constant $k \in \mathbb{Z}$
	$\mid 2^t$	accuracy embedding from Z to R
	$\mid t_1 + t_2 \mid t_1 - t_2$	integer arithmetic
	$\mid t_1 +\!\!+ t_2 \mid t_1 \times\!\!+ t_2 \mid t_1 -\!\!+ t_2 \mid t^{-\!\!+1}$	real arithmetic
	$\mid t_1 \mathbin{\hat{\leq}} t_2 \mid t_1 \mathbin{\hat{=}} t_2$	integer comparison
	$\mid t_1 \lesssim t_2$	real comparison
	$\mid \mathtt{choose}_n(t_1, \cdots, t_n)$	multivalued choice

ERC provides countably many nondeterminism constructs; i.e., for each natural number n greater than 1, there is $\mathtt{choose}_n$ which accepts n arguments.

For simplicity, we write $-t$ as an abbreviation for $0_{\mathsf{Z}} + (-t_2)$, $-t$ as an abbreviation for $0_{\mathsf{R}} + (-t)$, and t_1/t_2 as an abbreviation for $t_1 \times t_2{}^{-\!\!+1}$, $t_1 \gtrsim t_2$ as an abbreviation for $t_2 \lesssim t_1$, $t_1 \mathbin{\hat{\geq}} t_2$ as an abbreviation for $t_2 \mathbin{\hat{\leq}} t_1$.

In order to simplify presenting rules and definitions, we often write $\star$ to denote a symbol for the binary operations $\{+, \hat{\leq}, \hat{=}, -, -\!\!+, +, \times\!\!+, \lesssim\}$.

Commands and Programs in ERC

The commands in ERC are the commands in a simple imperative language:

c	$::=$ **skip**	skip
	$\mid x \coloneqq t$	variable assignment
	$\mid$ **var** $x : \tau \coloneqq t$	variable declaration
	$\mid c_1;\ c_2$	sequential composition
	$\mid$ **if** t **then** c_1 **else** c_2	branching
	$\mid$ **while** t **do** c	loop

The intended meaning of each construct is as follows. The construct **skip** is for the instruction of doing nothing. The construct $x \coloneqq t$ is to assign the value of t in the place of the variable x. The construct **var** $x : \tau \coloneqq t$ is to introduce a τ typed new variable named x with its value initialized to t. The construct **if** t **then** c_1 **else** c_2 is for ordinary branching instructions and **while** t **do** c is for ordinary while loops.

And, *programs* in ERC is of the form:

$$\begin{aligned}\mathcal{P} ::= \quad & \textbf{function}\ (x_1 : \tau_1, x_2 : \tau_2, \cdots, x_n : \tau_n) \\ & \quad c \\ & \textbf{return}\ t\end{aligned}$$

3.2.2 Typing Rules

The type system of ERC is given with typing rules for deriving well-typed terms, commands, and programs.

Well-typed Terms

We are not interested in any term. For example, due to the definition, $4 + \texttt{true}$ is a valid term in ERC. Instead of artificially giving meaning to those terms, we define well-typedness relation. And, those nonsense terms will be defined to be ill-typed.

A *context* Γ is a function from a finite set of variables to data types where $\cdot$ denotes the empty function. For a context Γ, a variable x not in $\mathrm{dom}(\Gamma)$, and a data type τ, we write $\Gamma, x{:}\tau$ to denote the context Γ extended with the mapping $x \mapsto \tau$. For two contexts Γ and Δ, when their domains are disjoint, we write Γ, Δ to denote the join of two functions.

Given a Γ, a term t, and a data type τ, we write $\Gamma \vdash t : \tau$ to say that t is judged to have type τ under the context Γ. It is defined inductively with the inference rules in Figure 3.1.

$$\frac{}{\Gamma \vdash k_{\mathsf{Z}} : \mathsf{Z}} \qquad \frac{}{\Gamma \vdash k_{\mathsf{R}} : \mathsf{R}} \qquad \frac{}{\Gamma \vdash \texttt{true} : \mathsf{K}} \qquad \frac{}{\Gamma \vdash \texttt{false} : \mathsf{K}} \qquad \frac{}{\Gamma \vdash \texttt{undef} : \mathsf{K}} \qquad \frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}$$

$$\frac{\Gamma \vdash t_1 : \mathsf{R} \qquad \Gamma \vdash t_2 : \mathsf{R}}{\Gamma \vdash t_1 \lesssim t_2 : \mathsf{K}} \qquad \frac{\Gamma \vdash t_1 : \mathsf{Z} \qquad \Gamma \vdash t_2 : \mathsf{Z}}{\Gamma \vdash t_1 \hat{\leq} t_2 : \mathsf{B}} \qquad \frac{\Gamma \vdash t_1 : \mathsf{Z} \qquad \Gamma \vdash t_2 : \mathsf{Z}}{\Gamma \vdash t_1 \hat{=} t_2 : \mathsf{K}}$$

$$\frac{\Gamma \vdash t_1 : \mathsf{Z} \qquad \Gamma \vdash t_2 : \mathsf{Z}}{\Gamma \vdash t_1 + t_2 : \mathsf{Z}} \qquad \frac{\Gamma \vdash t_1 : \mathsf{R} \qquad \Gamma \vdash t_2 : \mathsf{R}}{\Gamma \vdash t_1 + t_2 : \mathsf{R}} \qquad \frac{\Gamma \vdash t_1 : \mathsf{Z} \qquad \Gamma \vdash t_2 : \mathsf{Z}}{\Gamma \vdash t_1 - t_2 : \mathsf{Z}}$$

$$\frac{\Gamma \vdash t_1 : \mathsf{R} \qquad \Gamma \vdash t_2 : \mathsf{R}}{\Gamma \vdash t_1 - t_2 : \mathsf{R}} \qquad \frac{\Gamma \vdash t_1 : \mathsf{R} \qquad \Gamma \vdash t_2 : \mathsf{R}}{\Gamma \vdash t_1 \times t_2 : \mathsf{R}} \qquad \frac{\Gamma \vdash t : \mathsf{R}}{\Gamma \vdash t^{-\mathbf{1}} : \mathsf{R}}$$

$$\frac{\Gamma \vdash t_i : \mathsf{K} \quad (\text{for } i = 1, \cdots, n)}{\Gamma \vdash \texttt{choose}_n(t_1, \cdots, t_n) : \mathsf{Z}}\ n \geq 2 \qquad \frac{\Gamma \vdash t : \mathsf{Z}}{\Gamma \vdash 2^t : \mathsf{R}}$$

Figure 3.1: The typing rules for terms in ERC

See that for each integer $k \in \mathbb{Z}$, there is a constant k_{Z} of type Z and a constant k_{R} of type R. Though, syntactically k_{Z} and k_{R} are different terms of different types, when there is no ambiguity, when the well-typedness of a term containing k uniquely determines whether it is k_{Z} or k_{R}, we drop the subscript to simplify the presentation in the dissertation.

Well-typed Commands

As commands are constructed using terms, well-typedness also affects commands. Unlike terms, commands do not represent values but are meant to modify states. When we consider a command of the form of variable declaration $\mathbf{var}\ x : \tau \coloneqq t$, if the execution of it on some context Γ works well, it will create another context $\Gamma' \coloneqq \Gamma, x : \tau$. We write $\Gamma \vdash c \triangleright \Gamma'$ to denote that the command c is well-typed under the context Γ and executing it results in a new context Γ'. The well-typedness of commands are defined inductively using the inference rules in Figure 3.2.

$$\frac{}{\Gamma \vdash \mathbf{skip} \triangleright \Gamma} \qquad \frac{\Gamma \vdash t : \tau \quad \Gamma(x) = \tau}{\Gamma \vdash x \coloneqq t \triangleright \Gamma} \qquad \frac{x \notin \mathrm{dom}(\Gamma) \quad \Gamma \vdash t : \tau}{\Gamma \vdash \mathbf{var}\ \ x : \tau \coloneqq t\ \ \triangleright \Gamma, x : \tau} \qquad \frac{\Gamma \vdash c_1 \triangleright \Gamma_1 \quad \Gamma_1 \vdash c_2 \triangleright \Gamma_2}{\Gamma \vdash c_1; c_2 \triangleright \Gamma_2}$$

$$\frac{\Gamma \vdash t : \mathsf{K} \quad \Gamma \vdash c_1 \triangleright \Gamma \quad \Gamma \vdash c_2 \triangleright \Gamma}{\Gamma \vdash \mathbf{if}\ \ t\ \ \mathbf{then}\ \ c_1\ \ \mathbf{else}\ \ c_2\ \ \triangleright \Gamma} \qquad \frac{\Gamma \vdash t : \mathsf{K} \quad \Gamma \vdash c \triangleright \Gamma}{\Gamma \vdash \mathbf{while}\ \ t\ \ \mathbf{do}\ \ c\ \ \triangleright \Gamma}$$

Figure 3.2: The typing rules for commands in ERC

Notice that a new variable cannot be introduced in the body of a loop and the branches of a conditional statement.

Well-typed Programs

A program in ERC, which composes of a list of input variables, a command, and an output term, is well-typed if the command and the returned term are well-typed under the assumed input. We write $\vdash \mathcal{P} : \tau_1 \times \cdots \times \tau_n \to \tau$ to denote that a program $\mathcal{P}$ of the form

$$\begin{aligned} \mathcal{P} = \ & \mathbf{function}\ (x_1 : \tau_1, x_2 : \tau_2, \cdots, x_n : \tau_n) \\ & c \\ & \mathbf{return}\ t \end{aligned}$$

is judged to have type $\tau_1 \times \cdots \times \tau_n \to \tau$. It holds if and only if

$$x_1 : \tau_1, \cdots, x_n : \tau_n \vdash c \triangleright \Gamma \quad \text{and} \quad \Gamma \vdash t : \tau$$

hold for some Γ.

3.3 Denotational Semantics

Denotational semantics is a way to describe a program its mathematical meaning, abstracting its operational behavior away. For example, when we have a well-typed term $\Gamma \vdash t : \mathsf{R}$, we wish it to represent a real number as an element in $\mathbb{R}$. We express this interpretation formally by defining the denotational semantics of ERC. We begin with defining the most clear *denotations of data types*:

$$[\![\mathsf{Z}]\!] \coloneqq \mathbb{Z} \qquad [\![\mathsf{K}]\!] \coloneqq \mathbb{K} \qquad [\![\mathsf{R}]\!] \coloneqq \mathbb{R}\ .$$

The meaning of terms and commands get decided if a state is given. For example, the value of $x + y$ depends on the values that the variables x and y store. For a context Γ, a state in the context is a datum that records each variable's value. Formally speaking, the *denotation of a context* Γ is defined as follows:

$$[\![\Gamma]\!] \coloneqq \prod_{x \in \mathrm{dom}(\Gamma)} [\![\Gamma(x)]\!]$$

The denotation of a context is the set of *states* that are valid under the context. For a context $\gamma \in [\![\Gamma]\!]$, a variable x not in $\mathrm{dom}(\Gamma)$, and a value $v \in [\![\Gamma(\tau)]\!]$, we write $(\gamma, x \mapsto v)$ for γ extended with the assignment $x \mapsto v$. For two states $\gamma \in [\![\Gamma]\!]$ and $\delta \in [\![\Delta]\!]$ whose domains are disjoint, we write (γ, δ) to denote their joins. For a state $\gamma \in [\![\Gamma]\!]$, we write $\gamma[x \mapsto v]$ to denote the state γ whose assignment at x is updated with $x \mapsto v$.

3.3.1 Powerdomain for ERC

We are interested only in well-typed terms. Hence, we define the denotational semantics only for the well-typed terms. For a well-typed term $\Gamma \vdash t : \tau$, given a proper state $\gamma \in [\![\Gamma]\!]$, the term t is intended to represent a value in $[\![\tau]\!]$. The first attempt would be to make the denotation of the term be a function of type $[\![\Gamma]\!] \to [\![\tau]\!]$.

However, due to nondeterminism in ERC, a term may represent not a single value. For example, the well-typed term $\cdot \vdash \mathtt{choose}_2(\mathtt{true}, \mathtt{true}) : \mathsf{Z}$ is meant to represent 1 or 2 nondeterministically on the empty state. To make our denotational semantics capture the nondeterministic nature of ERC, we make it set-valued where the set denotes all possible nondeterministic outputs that the term evaluates to. Hence, in the above case, the denotation of $\cdot \vdash \mathtt{choose}_2(\mathtt{true}, \mathtt{true})$ is $\{1, 2\}$ since 1 and 2 are values that the term may evaluate nondeterministically.

In order to make the denotational semantics set-valued, we use Plotkin powerdomain which is devised to express Dijkstra's nondeterminism in [Plo76]:

Definition 3.1. On a flat domain $A_\perp \coloneqq A \cup \{\perp\}$, the *Plotkin powerdomain* $\mathbb{P}(A_\perp)$ is the set $\{S \subseteq A \mid S \neq \emptyset, S \text{ is finite or } \perp \in S\}$ endowed with the Egli-Milner ordering $\sqsubseteq$ characterized by

$$P \sqsubseteq Q \quad \iff \quad (\perp \in P \wedge P \subseteq Q \cup \{\perp\}) \vee (\perp \notin P \wedge P = Q) \ .$$

As the name suggests, it is a ω-CPO with a least element $\{\perp\}$.

Define the rectifying operation

$$A \supseteq S \mapsto S_\star \coloneqq \begin{cases} \{\perp\} & \text{if } S = \emptyset \ , \\ S & \text{if } S \text{ finite} \ , \\ S \cup \{\perp\} & \text{otherwise.} \end{cases}$$

The construction of powerdomains from flat domains as a mapping from a set A to the underlying set of $\mathbb{P}(A_\perp)$, is a monad [BVS93] where it on a function is defined by

$$\mathbb{P}((f : A \to B)_\perp) \coloneqq \bigcup_{x \in S} \begin{cases} \{f(x)\} & \text{if } x \neq \perp, \\ \{\perp\} & \text{otherwise.} \end{cases}$$

The unit is $\eta_A : x \mapsto \{x\}$ and the multiplication is

$$\mu_A : S \mapsto \bigcup_{T \in S} \begin{cases} T & \text{if } T \neq \perp, \\ \{\perp\} & \text{otherwise.} \end{cases}$$

Being a monad on Set, it admits coummative tensorial strength[1] $\beta_{A,B} : A \times \mathbb{P}(B_\perp) \to \mathbb{P}((A \times B)_\perp)$ which happens to be

$$\beta_{A,B}(x, S) = \bigcup_{y \in S} \begin{cases} \{(x, y)\} & \text{if } y \neq \perp \\ \{\perp\} & \text{otherwise.} \end{cases}$$

[1] Recall Section 2.4

And, it is lax monoidal where

$$\alpha_{A,B}(S,T) = \bigcup_{x\in S\wedge y\in T} \begin{cases} \{(x,y)\} & \text{if } x \neq \bot \wedge y \neq \bot, \\ \{\bot\} & \text{otherwise.} \end{cases}$$

generated by β.

Hence, we can define various types of lifting.

- When $f : A_1 \times \cdots \times A_d \to B$, we can lift it to $f^\dagger : \mathbb{P}((A_1)_\bot) \times \cdots \times \mathbb{P}((A_d)_\bot) \to \mathbb{P}(B_\bot)$ by consecutively precomposing appropriate α on $\mathbb{P}(f_\bot)$. It happens to be

$$f^\dagger(S_1, \cdots, S_d) = \bigcup_{(x_1,\cdots,x_d)\in S_1\times\cdots\times S_d} \begin{cases} \{f(x_1, \cdots, x_d)\} & \text{if } \forall i.\ x_i \neq \bot, \\ \{\bot\} & \text{otherwise.} \end{cases}$$

- When $f : A_1 \times \cdots \times A_d \to \mathbb{P}(B_\bot)$, we can lift it to $f^\dagger : \mathbb{P}((A_1)_\bot) \times \cdots \times \mathbb{P}((A_d)_\bot) \to \mathbb{P}(B_\bot)$ by consecutively precomposing appropriate α on $\mu_B \circ \mathbb{P}(f_\bot)$. It happens to be

$$f^\dagger(S_1, \cdots, S_d) = \bigcup_{(x_1,\cdots,x_d)\in S_1\times\cdots\times S_d} \begin{cases} f(x_1, \cdots, x_d) & \text{if } \forall i.\ x_i \neq \bot, \\ \{\bot\} & \text{otherwise.} \end{cases}$$

- When $f : A_1 \times \cdots \times A_d \to B$, we can lift it to $f^{\dagger_i} : A_1 \times \mathbb{P}((A_i)_\bot) \times \cdots \times A_d \to \mathbb{P}(B_\bot)$ by precomposing appropriate β on $\mathbb{P}(f_\bot)$. It happens to be

$$f^\dagger(x_1, \cdots, S_i, \cdots x_d) = \bigcup_{x_i\in S_i} \begin{cases} \{f(x_1, \cdots, x_i, \cdots x_d)\} & \text{if } x_i \neq \bot, \\ \{\bot\} & \text{otherwise.} \end{cases}$$

- When $f : A_1 \times \cdots \times A_d \to \mathbb{P}(B_\bot)$, we can lift it to $f^{\dagger_i} : A_1 \times \mathbb{P}((A_i)_\bot) \times \cdots \times A_d \to \mathbb{P}(B_\bot)$ by precomposing appropriate β on $\mu_B \circ \mathbb{P}(f_\bot)$. It happens to be

$$f^\dagger(x_1, \cdots, S_i, \cdots x_d) = \bigcup_{x_i\in S_i} \begin{cases} f(x_1, \cdots, x_i, \cdots x_d) & \text{if } x_i \neq \bot, \\ \{\bot\} & \text{otherwise.} \end{cases}$$

Thus far, though $\mathbb{P}(A_\bot)$ was a domain, we worked on only of its underlying set. The purpose was to simplify the presentation. However, as we need to use the domain-theoretic knowledge sooner or later, we need to show the operations revealed here satisfies the desired domain-theoretic conditions.

Lemma 3.1. The Kleisli composition is continuous in both arguments; i.e., for any $f_i, g : S \to \mathbb{P}(S_\bot)$ where $(f_i)_{i\in\mathbb{N}}$ is a chain,

$$\Big(\bigsqcup_{i\in\mathbb{N}} f_i\Big)^\dagger \circ g = \bigsqcup_{i\in\mathbb{N}} \big(f_i^\dagger \circ g\big) \quad \text{and} \quad g^\dagger \circ \bigsqcup_{i\in\mathbb{N}} f_i = \bigsqcup_{i\in\mathbb{N}} g^\dagger \circ f_i$$

hold.

Proof. The mappings forming chains are easy to see. Let $f = \bigsqcup_{i\in\mathbb{N}} f_i$. Suppose any $x \in S$. See that $\bot \in f^\dagger(g(x))$ if and only if $\bot \in g(x)$ or there is $y \in g(x)$ such that for all i, $\bot \in f_i(y)$.

Also, $\bot \in \bigsqcup_{i\in\mathbb{N}} \big(f_i^\dagger \circ g\big)(x)$ if and only if for all i, $\bot \in f_i^\dagger(g(x))$. See that this happens if and only if either $\bot \in g(x)$ or there is $y \in g(x)$ such that for all i, $\bot \in f_i(y)$.

Hence, $\bot \in f \circ g(x)$ if and only if $\bot \in \bigsqcup_{i\in\mathbb{N}} \big(f_i^\dagger \circ g\big)(x)$.

See that for any y such that $y \neq \bot$, $y \in f^\dagger(g(x))$ if and only if there is non bottom $z \in g(x)$ such that there is i where $y \in f(z)$.

And, for a non bottom y, $y \in \bigsqcup_{i\in\mathbb{N}} \left(f_i^\dagger \circ g\right)(x)$ if and only if there is i such that $y \in f_i^\dagger(g(x))$. This holds if and only if there is non bottom $z \in g(x)$ such that $y \in f_i(z)$.

Therefore, $f(x) = \bigsqcup_{i\in\mathbb{N}} \left(f_i^\dagger \circ g\right)(x)$ holds for all $x \in S$.

The other equation can be proven similarly.

□

For a mapping $f : A \to B_\bot$, let us define the *axiliary* codomain lifting $f^\ddagger : A \to \mathbb{P}(B_\bot)$ by

$$f^\ddagger(x) = \{f(x)\}.$$

3.3.2 Denotations of Terms

When A is the denotation of the type of a term t, the denotation of the term on a state will be defined as an element of the powerdomain $S \in \mathbb{P}(A_\bot)$. The case $\bot \in S$ denotes the case where the term is semantically ill-definite. Otherwise, S is the set of the values that t nondeterministically evaluates to.

The denotation of an expression is recursively defined naturally to the intended meaning of each operation lifted properly to the monad $\mathbb{P}(\square_\bot)$. Let us recall the definitions of some operations. The function $\lesssim$ for real comparison test is defined as follows:

$$\begin{array}{rcccl} \square_1 \lesssim \square_2 & : & \mathbb{R}\times\mathbb{R} & \to & \mathbb{K} \\ & \coloneqq & (x,y) & \mapsto & \begin{cases} tt & \text{if } x < y, \\ uk & \text{if } x = y, \\ ff & \text{if } x > y. \end{cases} \end{array}$$

The integer comparisons $\hat{=}, \hat{\leq}$ are functions to $\mathbb{K}$ such that the subset inclusion $\mathbb{2} \subseteq \mathbb{K}$ is postcomposed to the ordinary integer comparisons $=, \leq$.

Let us define operations that are not defined thus far. The multiplicative inversion $\square^{-\mathbf{1}}$ used here is a function to $\mathbb{R}_\bot$, $\bot$ extension of the partial mapping $x \mapsto x^{-1}$ that is not defined at $x = 0$. That is,

$$x^{-\mathbf{1}} = \begin{cases} x^{-1} & \text{if } x \neq 0, \\ \bot & \text{otherwise.} \end{cases}$$

And, define choose_n by

$$\begin{array}{rcccl} \mathsf{choose}_n & : & \mathbb{K}^n & \to & \mathbb{P}(\mathbb{Z}_\bot) \\ & \coloneqq & (b_1, \cdots, b_n) & \mapsto & \begin{cases} \{i \mid b_i = tt\} & \text{if } \exists i.\ b_i = tt, \\ \{\bot\} & \text{otherwise.} \end{cases} \end{array}$$

Given a well-typed term t such that $\Gamma \vdash t : \tau$, we define the function $[\![\Gamma \vdash t : \tau]\!] : [\![\Gamma]\!] \to \mathbb{P}[\![\tau]\!]_\bot$ recursively as in Fig. 3.3.

Though the definition of the denotational semantics looks a little complicated, considering the definition of the liftings, they are defined in a quite natural way. For example, the denotation $[\![\Gamma \vdash t_1 + t_2 : \mathsf{R}]\!]\gamma$ is defined to be

$$[\![\Gamma \vdash t_1 + t_2 : \mathsf{R}]\!]\gamma = \bigcup_{x\in[\![\Gamma\vdash t_1:\mathsf{R}]\!]\gamma, x\in[\![\Gamma\vdash t_1:\mathsf{R}]\!]\gamma} \begin{cases} \{x+y\} & \text{if } x \neq \bot \wedge y \neq \bot, \\ \{\bot\} & \text{otherwise.} \end{cases}$$

$$
\begin{aligned}
[\![\Gamma \vdash \texttt{true} : \mathsf{K}]\!]\gamma &\coloneqq \eta_{\mathbb{K}}(tt)\\
[\![\Gamma \vdash \texttt{false} : \mathsf{K}]\!]\gamma &\coloneqq \eta_{\mathbb{K}}(tt)\\
[\![\Gamma \vdash \texttt{undef} : \mathsf{K}]\!]\gamma &\coloneqq \eta_{\mathbb{K}}(uk)\\
[\![\Gamma \vdash k_{\mathsf{Z}} : \mathsf{Z}]\!]\gamma &\coloneqq \eta_{\mathbb{Z}}(k)\\
[\![\Gamma \vdash k_{\mathsf{R}} : \mathsf{R}]\!]\gamma &\coloneqq \eta_{\mathbb{R}}(k)\\
[\![\Gamma \vdash x : \tau]\!]\gamma &\coloneqq \eta_{[\![\tau]\!]}(\gamma(x))\\
[\![\Gamma \vdash t_1 \star t_2 : \tau]\!]\gamma &\coloneqq [\![\Gamma \vdash t_1 : \tau']\!]\gamma \star^{\dagger} [\![\Gamma \vdash t_2 : \tau']\!]\gamma \quad (\text{for } \star \in \{+, \hat{\lesssim}, \hat{=}, -, -, +, \times, \lesssim\})\\
[\![\Gamma \vdash t^{-\mathbf{1}} : \tau]\!]\gamma &\coloneqq ([\![\Gamma \vdash t : \tau']\!]\gamma)^{-1\downarrow^{\ddagger\dagger}_{\perp}}\\
[\![\Gamma \vdash \texttt{choose}_n(t_1, \cdots, t_n) : \mathsf{Z}]\!]\gamma &\coloneqq \texttt{choose}^{\dagger}_n([\![\Gamma \vdash t_1 : \mathsf{K}]\!]\gamma, \cdots, [\![\Gamma \vdash t_n : \mathsf{K}]\!]\gamma)
\end{aligned}
$$

Figure 3.3: The denotations of ERC terms.

That is, the denotation is the real number additions over all possible values of t_1 and t_2 with a condition that it includes $\perp$ if and only if $\perp$ is encountered.

Remark 3.1.

1. A term is semantically ill-defined when (i) its subterm is, (ii) we try to obtain the multiplicative inversion of zero, and (iii) there is no argument that `choose` can choose. See the ill-definiteness propagates in the sense that when the denotation of a subterm contains $\perp$, the term's denotation must contain $\perp$ as well.

2. Of course, the denotation of a real number comparison $x \lesssim y$ is partial in the sense that it contains uk when there is a real number r that is contained both in the denotation of x and the denotation of y.

3. The only possible type conversion from Z to R is done by $2^{\square} : n \mapsto 2^n$ called *accuracy embedding*.

4. An infinite set cannot be constructed with the definition. However, we use the powerdomain to be the domain of our denotational semantics since infinite sets become necessary in defining the denotations of commands.

3.3.3 Denotations of Commands

Since the intended meaning of a command is a state transformer, it is most natural to define the denotation of a well-typed command $[\![\Gamma \vdash S \triangleright \Gamma']\!]$ as a function of type $[\![\Gamma]\!] \to \mathbb{P}([\![\Gamma]\!]_{\perp})$ considering the nondeterminism in ERC. Given a state $\gamma \in [\![\Gamma]\!]$, executing S on γ will yields states in Γ' nondeterministically. Hence, intuitively, the denotation of S on γ is the set of all possible nondeterministic resulting states of executing S on γ. For example, the denotation of $x : \mathsf{Z} \vdash x \coloneqq \texttt{choose}(\texttt{true}, \texttt{true}) \triangleright x : \mathsf{Z}$ on a state $(x \mapsto 42)$ is $\{(x \mapsto 1), (x \mapsto 2)\}$.

What $\perp$ represents here is a little different. Semantical ill-definiteness still gets represented by $\perp$. For example, the denotation of $[\![\Gamma \vdash x \coloneqq t \triangleright \Gamma]\!]$ on a state γ contains $\perp$ if $[\![\Gamma \vdash t : \Gamma(x)]\!]\gamma$ contains $\perp$. Having loops and making the programming language expressive, infinite loops are always possible to

occur. For example, $\cdot \vdash \textbf{while } \texttt{true} \textbf{ do skip} \triangleright \cdot$ is a well-typed command, whose denotation has to be defined. In the case, we let the denotation of it contains $\bot$.

Define $\mathrm{Kond}_{\mathbb{P}(A_\bot)} : \mathbb{K} \times \mathbb{P}(A_\bot) \times \mathbb{P}(A_\bot) \to \mathbb{P}(A_\bot)$ by

$$\mathrm{Kond}_{\mathbb{P}(A_\bot)}(b, S, T) = \begin{cases} S & \text{if } b = tt, \\ T & \text{if } b = ff, \\ \{\bot\} & \text{otherwise.} \end{cases}$$

Let $\mathsf{LFP}_A(F)$ be the least fixed-point of $F : (A \to \mathbb{P}(A_\bot)) \to (A \to \mathbb{P}(A_\bot))$ when F is continuous with regards to the point-wise ordering. For any set S, maps $b : S \to \mathbb{P}(\mathbb{K}_\bot)$, and $c : S \to \mathbb{P}(S_\bot)$, define the map

$$\mathcal{W}_{b,c} : (f : S \to \mathbb{P}(S_\bot)) \mapsto \mathrm{Kond}^\dagger_{\mathbb{P}(A_\bot)} \circ (b \times (f^\dagger \circ c) \times \eta_S) \ .$$

Note that $\mathcal{W}_{b,S}$ is continuous by Lemma 3.1.

Given a well-typed command c such that $\Gamma \vdash c \triangleright \Gamma'$, we define the denotation $[\![\Gamma \vdash c \triangleright \Gamma']\!] : [\![\Gamma]\!] \to \mathbb{P}([\![\Gamma']\!]_\bot)$ recursively as in Fig. 3.4.

$$\begin{aligned}
[\![\Gamma \vdash \textbf{skip} \triangleright \Gamma]\!]\gamma &\coloneqq \eta_{[\![\Gamma]\!]}\gamma \\
[\![\Gamma \vdash x \coloneqq t \triangleright \Gamma]\!]\gamma &\coloneqq (v \mapsto \gamma[x \mapsto v])^\dagger \circ [\![\Gamma \vdash t : \tau]\!]\gamma \\
[\![\Gamma \vdash \textbf{var } x : \tau = t \triangleright \Gamma']\!]\gamma &\coloneqq (v \mapsto (\gamma, (x \mapsto v)))^\dagger \circ [\![\Gamma \vdash t : \tau]\!]\gamma \\
[\![\Gamma \vdash c_1; c_2 \triangleright \Gamma']\!]\gamma &\coloneqq [\![\Delta \vdash c_2 \triangleright \Gamma']\!]^\dagger \circ [\![\Gamma \vdash c_1 \triangleright \Delta]\!]\gamma \\
[\![\Gamma \vdash \textbf{if } t \textbf{ then } c_1 \textbf{ else } c_2 \triangleright \Gamma]\!]\gamma &\coloneqq \mathrm{Kond}^{\dagger_1}_{\mathbb{P}([\![\Gamma]\!]_\bot)}([\![\Gamma \vdash t : \mathsf{K}]\!]\gamma, [\![\Gamma \vdash c_1 \triangleright \Gamma]\!]\gamma, [\![\Gamma \vdash c_2 \triangleright \Gamma]\!]\gamma) \\
[\![\Gamma \vdash \textbf{while } t \textbf{ do } c \triangleright \Gamma]\!]\gamma &\coloneqq \mathsf{LFP}(\mathcal{W}_{[\![\Gamma \vdash t:\mathsf{K}]\!], [\![\Gamma \vdash c \triangleright \Gamma]\!]})
\end{aligned}$$

Figure 3.4: The denotations of ERC commands.

For most constructs, their denotations are defined naturally with regards to the lifting by the moand $\mathbb{P}(\square_\bot)$. Also, notice that $\bot$ propagates throughout its superterms. When a command S has a subterm or a subcommand that executes to $\bot$, the command executes to $\bot$. For example, when $[\![\Gamma \vdash b : \mathsf{K}]\!]\gamma$ contains $\bot$, the denotation $[\![\Gamma \vdash \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2 \triangleright \Gamma]\!]\gamma$ contains $\bot$. When $[\![\Gamma \vdash b : \mathsf{K}]\!]\gamma$ contains tt and $[\![\Gamma \vdash c_1 \triangleright \Gamma]\!]\gamma$ contains $\bot$, the denotation $[\![\Gamma \vdash \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2 \triangleright \Gamma]\!]\gamma$ contains $\bot$.

Let us put some comments on the while loops:

Remark 3.2.

1. The denotation of a while loop $\textbf{while } b \textbf{ do } S$ is defined in the way that it satisfies the recurrence equation:

$$[\![\Gamma \vdash \textbf{if } b \textbf{ then } c; (\textbf{while } b \textbf{ do } c) \textbf{ else skip} \triangleright \Gamma]\!] = [\![\Gamma \vdash \textbf{while } b \textbf{ do } c \triangleright \Gamma]\!].$$

2. Due to the fixed-point theorem, the denotation $[\![\Gamma \vdash \textbf{while } b \textbf{ do } c \triangleright \Gamma]\!]\gamma$ is the limit of the chain $\{\bot\} \sqsubseteq \mathcal{W}_{[\![\Gamma \vdash b:\mathsf{K}]\!], [\![\Gamma \vdash c \triangleright \Gamma]\!]}(\delta \mapsto \{\bot\})\gamma \sqsubseteq \mathcal{W}^2_{[\![\Gamma \vdash b:\mathsf{K}]\!], [\![\Gamma \vdash c \triangleright \Gamma]\!]}(\delta \mapsto \{\bot\})\gamma \sqsubseteq \cdots$.

3. The chain above can be seen as a possibly infinite sequence of unrolling the loop. When we define $A^{(m+1)}_{b,c} \coloneqq \textbf{if } b \textbf{ then } c; A^{(m)}_{b,c} \textbf{ else skip}$ and $A^{(0)}_{b,c} \coloneqq \textbf{while } \texttt{true} \textbf{ do skip}$, it holds that $\mathcal{W}^m_{[\![\Gamma \vdash b:\mathsf{K}]\!], [\![\Gamma \vdash c \triangleright \Gamma]\!]}(\delta \mapsto \{\bot\}) = [\![\Gamma \vdash A^{(m)}_{b,c} \triangleright \Gamma]\!]$ for all natural number m.

3.3.4 Denotations of Programs

Let the following ERC program

$$\begin{aligned}\mathcal{P} \coloneqq \textbf{function}\ &(x_1 : \tau_1, x_2 : \tau_2, \cdots, x_n : \tau_n)\\ &S\\ &\textbf{return}\ t\end{aligned}$$

be well-typed such that $\Gamma \vdash S \triangleright \Gamma'$ and $\Gamma' \vdash t : \tau$ for some context Γ' and data type τ where $\Gamma \coloneqq x_1 : \tau_1, x_2 : \tau_2, \cdots, x_n : \tau_n$. Then $\mathcal{P}$ denotes the function

$$[\![\mathcal{P}]\!] : [\![\tau_1]\!] \times \cdots \times [\![\tau_n]\!] \to \mathbb{P}([\![\tau]\!]_\bot)$$

defined by

$$[\![\mathcal{P}]\!](v_1, \cdots, v_n) \coloneqq [\![t]\!]^\dagger \circ [\![S]\!]((x_1 \mapsto v_1), \cdots, (x_n \mapsto v_n))$$

Consider a well-typed program

$$\mathcal{P} : \tau_1 \times \cdots \times \tau_d \to \tau.$$

For a partial multifunction $f :\subseteq [\![\tau_1]\!] \times \cdots \times [\![\tau_d]\!] \rightrightarrows [\![\tau]\!]$, we say the program $\mathcal{P}$ expresses f when

$$\forall (x_1, \cdots, x_d) \in \mathrm{dom}(f).\ \bot \notin [\![\mathcal{P}]\!](x_1, \cdots, x_d) \wedge [\![\mathcal{P}]\!](x_1, \cdots, x_d) \subseteq f(x_1, \cdots, x_d)$$

holds. For a partial function $f : [\![\tau_1]\!] \times \cdots \times [\![\tau_d]\!] \rightharpoonup [\![\tau]\!]$, we say the program $\mathcal{P}$ expresses f when it expresses the partial multifunction $(x_1, \cdots, x_d) \mapsto \{f(x_1, \cdots, x_d)\}$ that is defined on $\mathrm{dom}(f)$.

In a special case when $\tau = \mathsf{R}$ and $\tau_1 = \mathsf{Z}$, we say the program (approximately) expresses a partial function $f : [\![\tau_2]\!] \times \cdots \times [\![\tau_d]\!] \rightharpoonup \mathbb{R}$ when

$$\forall (x_1, \cdots, x_d) \in \mathrm{dom}(f).\ \forall p \in \mathbb{Z}.\ \forall y \in [\![\mathcal{P}]\!](x_1, \cdots, x_d).\ y \neq \bot \wedge |f(x_2, \cdots, x_d) - y| \leq 2^p$$

holds. In other words, seeing the first integer argument $x_1 : \mathsf{Z}$ as the precision parameter, the program computes 2^p approximation of $f(x_2, \cdots, x_d)$.

We conclude this section with the following completeness property of ERC:

Theorem 3.1 (Turing-Completeness over the Reals)**.** Every partial function $f : \mathbb{R} \rightharpoonup \mathbb{R}$ computable with regards to any effective representation of real numbers is expressible in ERC.

Proof. Consider a while programming language based on Peano arithmetic which provides integer multiplication. Let us call the language **while(PA)** and identify a program in the language with a partial function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ where for any natural number n, the program on n diverges if and only if $n \notin \mathrm{dom}(f)$ and returns m if and only if $m = f(n)$.

Our claim starts with that ERC can express any program in the language. When a program in **while(PA)** computes multiplication, for example, $x \coloneqq y \times z$, we can replace it with

$$\begin{aligned}&\textbf{var}\ x' : \mathsf{Z} \coloneqq 0_\mathsf{Z}; \textbf{var}\ y' : \mathsf{Z} \coloneqq y;\\ &\big(\textbf{while}\ y' \mathbin{\hat{\geq}} 1_\mathsf{Z}\ \textbf{do}\ x' \coloneqq x' + z;\ y' \coloneqq y' - 1_\mathsf{Z}\big);\\ &\big(\textbf{while}\ y' \mathbin{\hat{\leq}} -1_\mathsf{Z}\ \textbf{do}\ x' \coloneqq x' - z;\ y' \coloneqq y' + 1_\mathsf{Z}\big);\\ &x \coloneqq x'\end{aligned}$$

Of course, when a multiplication happens inside of a loop or the branches of a conditional statement, we need to declare the auxiliary variables in advance.

Since **while(PA)** is Turing-complete, ERC also is in the sense that for any computable partial function $f : \mathbb{N} \rightharpoonup \mathbb{N}$, there is a ERC program $\mathcal{P}_f : \mathsf{Z} \to \mathsf{Z}$ whose denotation restricted on $\mathbb{N}$ is f.

Now, consider **while(PA)** equipped with oracle. A program $P^?$ in the oracle while language **while(PA)**$^?$ is a program in **while(PA)** with the additional command construct

$$\mathsf{QUERY}(v, n).$$

When an oracle $\varphi \in \mathbb{N}^{\mathbb{N}}$ is equipped, the semantics of the above command is to assign $\varphi(n)$ at v when $n \geq 0$. The language **while(PA)**$^?$ is oracle Turing-complete.

By definition, (recall Section 2.5.2), a partial function $f : \mathbf{R}_{\text{dyadic}} \rightharpoonup \mathbf{R}_{\text{dyadic}}$ is computable if and only if there is a program $P^?$ in **while(PA)**$^?$ such that

$$\forall x \in \mathrm{dom}(f).\, \forall \varphi \in \mathbb{N}^{\mathbb{N}}.\, \left(\forall n.\, |x - \overline{\varphi(n)}/2^n| \leq 2^{-n}\right) \Rightarrow \forall n.\, |f(x) - \overline{P^{\varphi}(n)}/2^n| \leq 2^{-n}$$

holds. Here, $\overline{2k+1} = -k$ and $\overline{2k} = k$. Hence, by definition, for each computable partial function, there is a oracle program $P^?$ satisfying the above.

Now consider any computable partial function $f : \mathbb{R} \rightharpoonup \mathbb{R}$ and an oracle program $P^?$ whose input variable is p in **while(PA)**$^?$ corresponding to f. Suppose a ERC program whose input variables are $p : \mathsf{Z}$ and $x : \mathsf{R}$. In the very begeinning of the ERC program, we flip the sign of p by $p \coloneqq -p$. The body of the ERC program is that translated from $P^?$ by unrolling integer multiplications and translating each oracle query by

$$\mathsf{QUERY}(v, n) \implies \begin{array}{l} \text{//translate } n \text{ and store the value at } n' : \mathsf{Z} \\ x' : \mathsf{R} \coloneqq x \times 2^{n'}; \\ k : \mathsf{Z} \coloneqq 0; \\ \textbf{while } \mathtt{choose}_2(x' \lesssim 1, x' \gtrsim 1/2) \mathrel{\hat{=}} 2 \textbf{ do} \\ \quad k \coloneqq k + 1; \\ \quad x \coloneqq x' - 1 \\ \textbf{while } \mathtt{choose}_2(x' \gtrsim -1, x' \lesssim -1/2) \mathrel{\hat{=}} 2 \textbf{ do} \\ \quad k \coloneqq k - 1; \\ \quad x \coloneqq x' + 1; \\ \textbf{if } k \mathrel{\hat{\geq}} 0 \textbf{ then } v \coloneqq k + k \textbf{ else } v \coloneqq (-k) + (-k) + 1 \end{array}$$

See that for any real number $x \in \mathrm{dom}(f)$, the two commands do basically the same thing. Both assign a natural number $2k$ to v where $k/2^n$ is a 2^{-n} approximation of x or assign a natural number $2k+1$ to v where $-k/2^n$ is a 2^{-n} approximation of x.

At the last stage, suppose that the oracle program is returning a term t. We translate the term and store it to a variable $r : \mathsf{Z}$. We check if r is odd or even. If $r = k + k$ for some k, by repeatedly adding $1 : \mathsf{R}$, we obtain a real number variable $x : \mathsf{R}$ storing k. Otherwise, if $r = k + k + 1$ for some k, by repeated addition of $-1 : \mathsf{R}$, we obtain a real number variable $x : \mathsf{R}$ storing $-k$. We return the term $x \times 2^p$.

By the assumption, k is an integer such that $|f(x) - k2^p| \leq 2^p$. (since we flipped the sign of p). Hence, the returned term is 2^p approximation of $f(x)$ for any x. The translated program expresses f.

□

3.4 The Logic of ERC

Term evaluations being done precisely, reasoning on the behaviours of programs in ERC get simplified; there is no need to do tedious rounding-off analysis. For example, when we have a term t of an arithmetical expression, the term evaluates exactly to the value that t mathematically represents. The goal is to make a framework where we can use the mathematical structure $\mathcal{S}_{\mathrm{ERC}}$ of Presburger arithmetic, real closed field, and Kleene logic, to reason the properties of programs in ERC .

Nonetheless, there are terms in ERC which are not arithmetical. For example, $\texttt{choose}(t_1, t_2, \cdots, t_n)$, the construct generating nondeterminism, is not an arithmetical expression; i.e., there is no function in the mathematical structure that corresponds to `choose`. Hence, it is non-trivial how to use the mathematical structures $\mathcal{S}_{\mathrm{ERC}}$ to describe and reason about the behaviours of ERC programs.

The task is divided into two tasks at different levels. The first is to show how we can use the structure $\mathcal{S}_{\mathrm{ERC}}$ to *describe* the properties of ERC programs. For the task, we study the structure and define a logical language $\mathcal{L}_{\mathrm{ERC}}$, which is the first-order logic on $\mathcal{S}_{\mathrm{ERC}}$ in Section 3.4.1. In Section 3.4.2, we show how the property of a program in ERC can be described using $\mathcal{L}_{\mathrm{ERC}}$. We define *spcifications* formally. The next task is to devise a framework where we can reason on the behaviours of programs. When we have a specification written for a program, we should either accept the specification by proving it or reject the specification by disproving it.

3.4.1 Assertion Language $\mathcal{L}$

Definition 3.2. The *Structure of* ERC $\mathcal{S}$ is the three-sorted structure combining the Kleene logic $(\mathbb{K}, \mathit{ff}, \mathit{tt}, \mathit{uk})$ with Presburger arithmetic $(\mathbb{Z}, 0, 1, +, -, \leq, 2\mathbb{Z}, 3\mathbb{Z}, 4\mathbb{Z}, \ldots)$ and ordered field $(\mathbb{R}, 0, 1, +, -, \times, <)$. They are connected via the *binary accuracy* mapping $2^{\square} : \mathbb{Z} \ni p \mapsto 2^p \in \mathbb{R}$ and its partial half-inverse $\lfloor \log_2 \circ \mathrm{abs} \rfloor : \mathbb{R} \setminus \{0\} \to \mathbb{Z}$. Here $k\mathbb{Z}$ denotes the predicate on $\mathbb{Z}$ which is precisely all integer multiples of $k \in \mathbb{N}$. The *Logic of* ERC $\mathcal{L}$ is the first-order language of $\mathcal{S}$; the *Theory of* ERC $\mathcal{T}$ is the complete first-order theory of the structure. Of course, the equality predicate is implicitly included in each sort. □

It should not be confused with the informal language which we used to define the denotational semantics in 3.3. For example, $\bot$ is a symbol that does not appear in $\mathcal{S}$. Also, the predicates $\leq, <, =$ above are logical predicates that are not the functions $\hat{\leq}, \hat{=}, \lesssim$ to $\mathbb{K}$.

We say a formula is well-formed under an ERC context Γ if it is a well-formed formula in $\mathcal{L}$ when each variable x in $\mathrm{dom}(\Gamma)$ is of the sort $[\![\Gamma(x)]\!]$. We write $\Gamma \Vdash \phi$ to say that ϕ is a well-formed formula under Γ and $\mathsf{wf}(\Gamma)$ to be the set of well-formed formulae under Γ. Similarly, for an assignment $\gamma \in [\![\Gamma]\!]$, we write $\gamma \vDash \phi$ to say that γ validates ϕ under the standard interpretation. We define the *semantics of a well-formed formula* to be $[\![\Gamma \Vdash \phi]\!] \coloneqq \{\gamma \in [\![\Gamma]\!] \mid \gamma \vDash \phi\}$ the set of assignments that validate ϕ.

The following lemma shows that $\mathcal{L}$ is expressive enough to express the term language of ERC .

Lemma 3.2. The logic of ERC is expressive for the term language. To each well-typed term $\Gamma \vdash t : \tau$ and variable $y \notin \mathrm{dom}(\Gamma)$, there is a well-formed formula $\Gamma, y{:}\tau \Vdash (\!|\Gamma \vdash t : \tau|\!)_y$ such that

$$(\gamma, y \mapsto v) \vDash (\!|\Gamma \vdash t : \tau|\!)_y \text{ if and only if } v \in [\![\Gamma \vdash t : \tau]\!]\gamma \text{ and } \bot \notin [\![\Gamma \vdash t : \tau]\!]\gamma$$

holds. In other words, if there is $(\gamma, y \mapsto v)$ that validates $(\!|\Gamma \vdash t : \tau|\!)_y$, it implies that the term is semantically well-defined under γ and the denotation contains v. For the opposite direction, if the term is semantically well-defined having v in its denotation, $\gamma, y \mapsto v \vDash (\!|\Gamma \vdash t : \tau|\!)_y$ holds.

Proof. Suppose a function $f : A_1 \times \cdots \times A_d \to \mathbb{P}(B_\perp)$ where $A_i, B \in \{\mathbb{K}, \mathbb{Z}, \mathbb{R}\}$ is definable in the sense that there is a well-formed formula $x_1 : A_1, \cdots, x_d : A_d, y : B \Vdash (\!|f|\!)$ such that for any $\gamma \in \Pi_{x_i} A_i$ and $v \in B$,

$$(\gamma, y \mapsto v) \vDash (\!|f|\!) \Leftrightarrow \perp \notin f(\gamma(x_1), \cdots, \gamma(x_d)) \wedge v \in f(\gamma(x_1), \cdots, \gamma(x_d)) \ .$$

And, suppose $S_i : A_1 \times \cdots \times A_d \to \mathbb{P}((A_i)_\perp)$ is definable by $(\!|S_i|\!)$ as the above sense. Then, $f^\dagger \circ (S_1 \times \cdots \times S_d)$ is definable as well by

$$(\!|f^\dagger \circ (S_1 \times \cdots \times S_d)|\!)(y) = \exists y_1, \cdots, y_d. \ (\!|f|\!) \wedge (\!|S_1|\!)[y_1/y] \wedge \cdots \wedge (\!|S_d|\!)[y_d/y].$$

Suppose $f : A_1 \times \cdots \times A_d \to B$ where $A_i, B \in \{\mathbb{K}, \mathbb{Z}, \mathbb{R}\}$ is definable in that there is a well-formed formula $x_1 : A_1 \cdots, x_d : A_d, y : B \Vdash (\!|f|\!)$ such that for any $\gamma \in \Pi_{x_i} A_i$ and $v \in B$

$$(\gamma, y \mapsto v) \vDash (\!|f|\!) \Leftrightarrow v = f(\gamma(x_1), \cdots, \gamma(x_d)) \ .$$

And, suppose $S_i : A_1 \times \cdots \times A_d \to \mathbb{P}((A_i)_\perp)$ is definable by $(\!|S_i|\!)$. Then, $f^\dagger \circ (S_1 \times \cdots \times S_d)$ is definable as well since $\eta_B \circ f$ is definable by the same formula $(\!|f|\!)$.

See that all atomic operations are definable. For example,

$$(\!|\mathsf{choose}_n|\!)(b_1, \cdots, b_n, b) = (b = 1 \wedge b_1 = tt) \vee \cdots (b = n \wedge b_n = tt),$$

$$(\!|\lesssim|\!)(x, y, b) = (b = tt \wedge x < y) \vee (b = uk \wedge x = y) \vee (b = ff \wedge y < x),$$

and

$$(\!|\Box^{-1|_\perp^\ddagger}|\!)(x, y) = (y \times x = 1) \ .$$

Hence, the denotation of the term language is definable as the denotation of a term is defined by Kleisli compositions of atomic operations. □

For a well-typed term $\Gamma \vdash t : \tau$, and for any term x in $\mathcal{L}$, let us write $(\!|\Gamma \vdash t : \tau|\!)(x)$ to denote $(\!|\Gamma \vdash t : \tau|\!)_y[x/y]$.

The following theorem shows that $\mathcal{T}$ is decidable; every first-order sentence in $\mathcal{L}$ can be formally either verified or refuted. This applies, for example, to pre/post conditions or loop invariants of ERC programs. It differs significantly from traditional programming languages for discrete data: Recall that, for example, classical `WHILE` programs over integers with multiplication do suffer from Gödel undecidability [Coo78, §6].

Theorem 3.2 (Decidability of the Logic of ERC)**.**

a) The Theory $\mathcal{T}$ of ERC is decidable.

b) It is also 'model complete' in that it admits elimination of quantifiers up to one (either existential or universal) block.

c) However, replacing $2^\Box$ with the 'unary accuracy' embedding $\mathbb{N}_+ \ni n \mapsto 1/n \in \mathbb{R}$ destroys decidability.

Proof. c) Including a unary predicate $\mathbb{Z}$, or (any total extension of) the unary accuracy embedding, allows to express integer multiplication via the reals, since $m \times n = /\big((1/m) \times \jmath(1/n)\big)$ hence recovers Gödel undecidability via Robinson's Theorem.

a)+b) A celebrated result of van den Dries [Dri86] asserts quantifier elimination for the expanded first-order theory of real-closed fields

$$(\mathbb{R}, 0, 1, +, -, \times, <, 2^{k\mathbb{Z}} : k \in \mathbb{N}, 2^{\lfloor \log_2 \circ \mathrm{abs} \rfloor}) \tag{3.1}$$

with axiomatized additional predicates $2^{k\mathbb{Z}}$, $k \in \mathbb{N}$, and truncation function to binary powers $2^{\lfloor \log_2 \circ \mathrm{abs} \rfloor}$, see also [AY07].

Note that both the real-closed field $(\mathbb{R}, 0, 1, +, -, \times, <)$ and Presburger Arithmetic can be embedded into the expanded structure from Equation (3.1); the latter interpreted as its multiplicative variant $(2^{\mathbb{Z}}, 1, 2, \times, <, 2^{k\mathbb{Z}} : k \in \mathbb{N})$ is called *Skolem Arithmetic* [Bés02b]:

- Replace quantifiers over Skolem integers with real quantifiers subject to the predicate $2^{k\mathbb{Z}}$ for $k \coloneqq 1$;
- Consider $2^{\square} : \mathbb{Z} \to \mathbb{R}$ as the restricted identity $\mathrm{id}_{2^{\mathbb{Z}}}$ in $\mathbb{R}$.

Then every formula φ with or without parameters in our two-sorted structure translates signature by signature to an equivalent one $\tilde{\varphi}$ over the expanded theory where quantifiers can be eliminated, yielding equivalent decidable $\tilde{\psi}$ (which may involve binary truncation $2^{\lfloor \log_2 \circ \mathrm{abs} \rfloor}$).

To translate this back to some equivalent ψ over the two-sorted structure, while re-introducing only one type of quantifiers, observe that for real x:

$$\begin{aligned} x \in 2^{k\mathbb{Z}} &\quad\Leftrightarrow\quad \exists z \in \mathbb{Z}.\, z \in k\mathbb{Z} \;\wedge\; x = 2^z; \\ x \notin 2^{k\mathbb{Z}} &\quad\Leftrightarrow\quad \exists z \in \mathbb{Z}.\, z \in k\mathbb{Z} \;\wedge\; 2^z < x < 2^{z+k}\,. \end{aligned}$$

Similarly, replace real binary truncation $2^{\lfloor \log_2 \circ \mathrm{abs}(x) \rfloor}$ with "2^z" for some/every $z \in \mathbb{Z}$ s.t. $2^z \leq |x| < 2^z + 1$ in case $x > 0$, with 0 otherwise.

The Kleene Algebra $\mathbb{K}$ is finite and does not affect decidability. □

3.4.2 Reasoning Principles

We use precondition-postcondition-style program specifications. We use the specification language $\mathcal{L}$ to specify the properties of a program. Recall that the denotation of a well-typed command $[\![\Gamma \vdash c \triangleright \Gamma']\!]$ is s state transformer. Given an initial state $\gamma \in [\![\Gamma]\!]$, it gives us a set $[\![\Gamma \vdash c \triangleright \Gamma']\!]\gamma$. It contains $\bot$ when the command is semantically ill-defined: i.e., if there is a nondeterministic branch in the execution of S that results in an error. Otherwise, it is the set of all resulting states that the nondeterministic branches in the execution of c yield.

Considering that the semantics of a well-formed formula $\Gamma \vdash \phi$ in $\mathcal{L}$ is a subset of states $[\![\Gamma]\!]$, we can use formulae to describe the behaviour of a command. Informally, we can pick two formulae ϕ and ψ where ϕ is well-formed under Γ and ψ is well-formed under Γ' to say that for any execution of c under a state in $[\![\Gamma \Vdash \phi]\!]$ results states in $[\![\Gamma' \Vdash \psi]\!]$. In this scenario, the formula ϕ is a *precondition* and the formula ψ is a *postcondition*. We can formalize this as follows:

Definition 3.3. A (total correctness) specification of a well-typed command $\Gamma \vdash c \triangleright \Gamma'$ is of the form $\Gamma \vdash \left[\phi\right]\ c\ \left[\psi\right] \triangleright \Gamma'$ where $\phi \in \mathsf{wf}(\Gamma)$ and $\psi \in \mathsf{wf}(\Gamma')$. Its meaning is that for any initial state satisfying $\gamma \in [\![\Gamma \Vdash \phi]\!]$, the command is semantically well-defined $\bot \notin [\![\Gamma \vdash S \triangleright \Gamma']\!]\gamma$ and each resulting state $\gamma' \in [\![\Gamma \vdash c \triangleright \Gamma']\!]\gamma$ satisfies $\gamma' \in [\![\Gamma' \Vdash \psi]\!]$.

We use specifications to describe the property of a command. And, we need a tool to reason on if the specification is correct. We devise proof rules, an axiomatic semantics, to enable reasoning

on specifications without referring to the denotational semantics. Let us simplify $[\![\Gamma \vdash t : \tau]\!]$ to $[\![t]\!]$, $[\![\Gamma \vdash c \triangleright \Gamma']\!]$ to $[\![c]\!]$, and $(\!|\Gamma \vdash t : \tau|\!)$ to $(\!|t|\!)$ when it is obvious what the omitted items are from the context.

We take Hoare-style proof rules to derive correct specifications that are defined as follows.

Definition 3.4. The verification calculus of ERC is a formal system which consists of the proof rules and axioms for deriving correct specifications defined in Figure 3.5.

$$\frac{\Gamma \vdash \left[\phi'\right]\ c\ \left[\psi'\right] \triangleright \Gamma'}{\Gamma \vdash \left[\phi\right]\ c\ \left[\psi\right] \triangleright \Gamma'}\ \phi \Rightarrow \phi' \text{ and } \psi' \Rightarrow \psi \qquad \frac{}{\Gamma \vdash \left[\psi\right]\ \mathbf{skip}\ \left[\psi\right] \triangleright \Gamma}$$

$$\frac{}{\Gamma \vdash \left[(\exists y\,.\,(\!|t|\!)(y)) \wedge \forall y\,.\,(\!|t|\!)(y) \Rightarrow \psi[y/x]\right]\ x \coloneqq t\ \left[\psi\right] \triangleright \Gamma}$$

$$\frac{}{\Gamma \vdash \left[(\exists y.\ (\!|t|\!)(y)) \wedge \forall y.\ (\!|t|\!)(y) \Rightarrow \psi[y/x]\right]\ \mathbf{var}\ \ x : \tau \coloneqq t\ \left[\psi\right] \triangleright \Gamma, x : \tau}$$

$$\frac{\Gamma \vdash \left[\phi\right]\ c_1\ \left[\theta\right] \triangleright \Gamma_1 \quad \Gamma_1 \vdash \left[\theta\right]\ c_2\ \left[\psi\right] \triangleright \Gamma_2}{\Gamma \vdash \left[\phi\right]\ c_1; c_2\ \left[\psi\right] \triangleright \Gamma_2}$$

$$\frac{\Gamma \vdash \left[\phi \wedge\ (\!|t|\!)(tt)\right]\ c_1\ \left[\psi\right] \triangleright \Gamma \quad \Gamma \vdash \left[\phi \wedge (\!|t|\!)(\mathit{ff})\right]\ c_2\ \left[\psi\right] \triangleright \Gamma}{\Gamma \vdash \left[\phi \wedge ((\!|t|\!)(tt) \vee (\!|t|\!)(\mathit{ff})) \wedge \neg(\!|t|\!)(uk)\right]\ \mathbf{if}\ \ t\ \ \mathbf{then}\ \ c_1\ \ \mathbf{else}\ \ c_2\left[\psi\right] \triangleright \Gamma}$$

$$\frac{\Gamma, \xi : \mathsf{R}, \xi' : \mathsf{R} \vdash \left[(\!|t|\!)(tt) \wedge I \wedge V = \xi \wedge L = \xi'\right]\ c\ \left[I \wedge V \leq \xi - \xi' \wedge L = \xi'\right] \triangleright \Gamma, \xi : \mathsf{R}, \xi' : \mathsf{R}}{\Gamma \vdash \left[I\right]\ \mathbf{while}\ \ t\ \ \mathbf{do}\ \ c\left[I \wedge (\!|t|\!)(\mathit{ff})\right] \triangleright \Gamma}$$

The rule for while loop has the side-conditions:

- $I \wedge (\!|t|\!)(tt) \Rightarrow L > 0$
- $I \Rightarrow ((\!|t|\!)(tt) \vee (\!|t|\!)(\mathit{ff})) \wedge \neg(\!|b|\!)(uk)$
- $I \wedge V \leq 0 \Rightarrow \forall k.\ (\!|t|\!)(k) \Rightarrow k = \mathit{ff}$
- ξ, ξ' does not appear free in I, V, L

Figure 3.5: The verification calculus of ERC .

Let us put some remarks:

Remark 3.3.

1. When we execute $x \coloneqq t$ on a state $\gamma \in [\![\Gamma]\!]$, any correct precondition should ensure that $\bot \notin [\![x]\!]\gamma$ which is precisely when $\exists y\,.\,(\!|x|\!)(y)$ holds. The postcondition ψ holds after after replacing x with for any $y \in [\![t]\!]\gamma$. The values in $[\![t]\!]\gamma$ are defined by y that satisfies $(\!|t|\!)(y)$. Hence, when γ satisfies $\forall y.\ (\!|t|\!)(y) \Rightarrow \psi[y/x]$, it holds that $\gamma[x \mapsto y]$ satisfy ψ for any $y \in [\![t]\!]\gamma$.
2. (Conditional) When a state γ validates $(\!|t|\!)(tt)$, we can guarantee two things: $\bot \notin [\![t]\!]\gamma$ and $tt \in [\![t]\!]\gamma$. Similarly, γ satisfying $(\!|t|\!)(\mathit{ff})$ ensures that $\bot \notin [\![t]\!]\gamma$ and $\mathit{ff} \in [\![t]\!]\gamma$; and γ satisfying $(\!|t|\!)(uk)$ ensures

that $\bot \notin [\![t]\!]\gamma$ and $uk \in [\![t]\!]\gamma$. Therefore, γ satisfying $((\!|t|\!)(tt) \vee (\!|t|\!)(\mathit{ff})) \wedge \neg(\!|t|\!)(uk)$ ensures that $[\![t]\!]\gamma$ is either one of $\{tt\}$, $\{\mathit{ff}\}$, or $\{tt, \mathit{ff}\}$.

For any state γ that makes the first branch to be taken, it satisfies $(\!|t|\!)(tt)$. Hence, by the first premise, the execution satisfies ψ, similarly for the second branch.

3. (While) The formula I is a loop invariant, the term V is a loop variant, and the term L is a lower bound on decrement of V. The premise ensures that I is indeed a loop invariant and V is a quantity that decreases throughout iterations by at least L, which is a positive invariant quantity throughout iterations. The side-conditions ensure that as long as I holds, the evaluation of t is never uk, when V gets less than or equal to zero, the evaluation of t must be ff.

   The variables ξ, ξ' does not appear in Γ; they are so-called *ghost variables*.

Having a formal system, the question arises on whether it is sound and complete. Our Hoare logic is sound but not complete. The remaining of this subsection is about the issues.

Proof of the Soundness

Let us start with the statement:

Theorem 3.3. The verification calculus of ERC is sound. In other words, if $\Gamma \vdash c \triangleright \Gamma'$ is derivable by the proof rules in Definition 3.4, its meaning according to Definition 3.3 holds.

We start the proof with proving the lemma which is a characterization of the denotations of while loops:

Lemma 3.3. For a well-typed command $\Gamma \vdash \textbf{while}\ \ t\ \ \textbf{do}\ \ c\ \triangleright \Gamma$, define the sequences of set-valued functions on $[\![\Gamma]\!]$:

- $B^0_{t,c}\gamma \coloneqq \{\gamma\}$
- $C^0_{t,c}\gamma \coloneqq \emptyset$
- $B^{n+1}_{t,c}\gamma \coloneqq \bigcup_{\delta \in B^n_{t,c}\gamma} \bigcup_{\substack{l \in [\![t]\!]\delta \\ \delta' \in [\![S]\!]\delta}} \begin{cases} \{\delta'\} & \text{if } l = tt \wedge \delta' \neq \bot, \\ \emptyset & \text{otherwise.} \end{cases}$
- $C^{n+1}_{t,c}\gamma \coloneqq C^n_{t,c}\gamma \cup \bigcup_{\delta \in B^n_{t,c}\gamma} \bigcup_{\substack{l \in [\![b]\!]\gamma \\ \delta' \in [\![c]\!]\delta}} \begin{cases} \{\delta\} & \text{if } l = \mathit{ff}, \\ \emptyset & \text{if } l = tt \wedge \delta' \neq \bot, \\ \{\bot\} & \text{otherwise.} \end{cases}$

Then, for all $n \in \mathbb{N}$, it holds that $[\![A^n_{t,c}]\!]\gamma = C^n_{t,c}\gamma \cup \{\bot \mid \exists x \in B^n_{t,c}\gamma\}$.

Intuitively, $B^n_{t,c}\gamma$ is the set of states that requires further execution after running the while loop on γ for n times. $C^n_{t,c}\gamma$ is the set of states that have escaped from the loop (either because ff has been evaluated or $\bot$ has occurred) during running the loop for n times.

Proof. Let us drop the subscripts t, c for the convenience of the presentation. We first prove the following alternative characterization of the sequence of sets:

$$B^{n+1}\gamma = \bigcup_{\substack{\ell \in [\![t]\!]\gamma \\ \delta \in [\![c]\!]\gamma}} \begin{cases} B^n(\delta) & \text{if } \ell = tt \wedge \delta \neq \bot, \\ \emptyset & \text{otherwise.} \end{cases}$$

It is trivial when $n = 0$. Now, suppose the equation holds for all γ and for all n up to m. Then the following derivation shows that the characterization is valid for $n = m + 1$ as well.

$$
\begin{aligned}
B^{m+2}\gamma &= \bigcup_{\gamma \in B^{m+1}\gamma} \bigcup_{\substack{\ell \in [\![t]\!]\gamma \\ \delta \in [\![c]\!]\gamma}} \begin{cases} \{\delta\} & \text{if } \ell = tt \wedge \delta \neq \bot, \\ \emptyset & \text{otherwise.} \end{cases} \\
&= \bigcup_{\gamma \in \bigcup_{\substack{\ell' \in [\![t]\!]\gamma \\ \delta' \in [\![c]\!]\gamma}} \begin{cases} B^m(\delta') & \text{if } \ell' = tt \wedge \delta' \neq \bot, \\ \emptyset & \text{otherwise.} \end{cases}} \bigcup_{\substack{\ell \in [\![t]\!]\gamma \\ \delta \in [\![c]\!]\gamma}} \begin{cases} \{\delta\} & \text{if } \ell = tt \wedge \delta \neq \bot, \\ \emptyset & \text{otherwise.} \end{cases} \\
&= \bigcup_{\substack{\ell' \in [\![t]\!]\gamma \\ \delta' \in [\![c]\!]\gamma}} \begin{cases} \bigcup_{\gamma \in B^m(\delta')} \bigcup_{\substack{\ell \in [\![t]\!]\gamma \\ \delta \in [\![c]\!]\gamma}} \begin{cases} \{\delta\} & \text{if } \ell = tt \wedge \delta \neq \bot, \\ \emptyset & \text{otherwise.} \end{cases} & \text{if } \ell' = tt \wedge \delta' \neq \bot, \\ \emptyset & \text{otherwise.} \end{cases} \\
&= \bigcup_{\substack{\ell' \in [\![t]\!]\gamma \\ \delta' \in [\![c]\!]\gamma}} \begin{cases} B^{m+1}(\delta') & \text{if } \ell' = tt \wedge \delta' \neq \bot, \\ \emptyset & \text{otherwise.} \end{cases}
\end{aligned}
$$

We now show the following characterization:

$$
C^{n+1}\gamma = \bigcup_{\substack{\ell \in [\![t]\!]\gamma \\ \delta \in [\![c]\!]\gamma}} \begin{cases} C^n(\delta) & \text{if } \ell = tt \wedge \delta \neq \bot, \\ \{\gamma\} & \text{if } \ell = \mathit{ff}, \\ \{\bot\} & \text{otherwise.} \end{cases}
$$

It is easy to show that the equation holds for $n = 0$. Now, assume the equation holds for all n up to m. Then,

$$
\begin{aligned}
C^{m+2}\gamma &= C^{m+1}\gamma \cup \bigcup_{\delta \in B^{m+1}\gamma} \bigcup_{\substack{\ell' \in [\![t]\!]\delta \\ \delta' \in [\![c]\!]\delta}} \begin{cases} \{\delta\} & \text{if } \ell' = \mathit{ff}, \\ \emptyset & \text{if } \ell' = tt \wedge \delta' \neq \bot, \\ \{\bot\} & \text{otherwise.} \end{cases} \\
&= C^{m+1}\gamma \cup \bigcup_{\delta \in \bigcup_{\substack{\ell \in [\![t]\!]\gamma \\ \gamma \in [\![c]\!]\gamma}} \begin{cases} B^m\gamma & \text{if } \ell = tt \wedge \gamma \neq \bot, \\ \emptyset & \text{otherwise.} \end{cases}} \bigcup_{\substack{\ell' \in [\![t]\!]\delta \\ \delta' \in [\![c]\!]\delta}} \begin{cases} \{\delta\} & \text{if } \ell' = \mathit{ff}, \\ \emptyset & \text{if } \ell' = tt \wedge \delta' \neq \bot, \\ \{\bot\} & \text{otherwise.} \end{cases} \\
&= C^{m+1}\gamma \cup \bigcup_{\substack{\ell \in [\![t]\!]\gamma \\ \gamma \in [\![c]\!]\gamma}} \begin{cases} \bigcup_{\delta \in B^m\gamma} \bigcup_{\substack{\ell \in [\![t]\!]\delta \\ \delta' \in [\![c]\!]\delta}} \begin{cases} \{\delta\} & \text{if } \ell' = \mathit{ff} \\ \emptyset & \text{if } \ell' = tt \wedge \delta' \neq \bot, \\ \{\bot\} & \text{otherwise.} \end{cases} & \text{if } \ell = tt \wedge \gamma \neq \bot, \\ \emptyset & \text{otherwise.} \end{cases}
\end{aligned}
$$

$$= \bigcup_{\substack{\ell\in[\![t]\!]\gamma\\ \gamma\in[\![c]\!]\gamma}} \begin{cases} C^m\gamma \cup \bigcup_{\delta\in B^m\gamma} \bigcup_{\substack{\ell\in[\![t]\!]\delta\\ \delta'\in[\![c]\!]\delta}} \begin{cases} \{\delta\} & \text{if } \ell' = \mathit{ff} \\ \emptyset & \text{if } \ell' = \mathit{tt} \wedge \delta' \neq \bot, \\ \{\bot\} & \text{otherwise.} \end{cases} & \text{if } \ell = \mathit{tt} \wedge \gamma \neq \bot, \\ \{\gamma\} & \text{if } \ell = \mathit{ff}, \\ \{\bot\} & \text{otherwise.} \end{cases}$$

$$= \bigcup_{\substack{\ell\in[\![t]\!]\gamma\\ \gamma\in[\![c]\!]\gamma}} \begin{cases} C^{m+1}\gamma & \text{if } \ell = \mathit{tt} \wedge \gamma \neq \bot, \\ \{\gamma\} & \text{if } \ell = \mathit{ff}, \\ \{\bot\} & \text{otherwise.} \end{cases}$$

Now, using the suggested characterization, we prove $[\![A^n_{t,c}]\!]\gamma = C^n_{t,c}\gamma \cup \{\bot \mid \exists x \in B^n_{t,c}\gamma\}$ for all $n \in \mathbb{N}$. When $n = 0$, both are $\{\bot\}$. Suppose the equation holds for $n = m$. Then,

$$[\![A^{m+1}]\!]\gamma = \bigcup_{\substack{\ell\in[\![t]\!]\gamma\\ \delta\in[\![c]\!]\gamma}} \begin{cases} [\![A^m]\!]\delta & \text{if } \ell = \mathit{tt} \wedge \delta \neq \bot, \\ \{\gamma\} & \text{if } \ell = \mathit{ff}, \\ \{\bot\} & \text{otherwise.} \end{cases}$$

$$= \bigcup_{\substack{\ell\in[\![t]\!]\gamma\\ \delta\in[\![c]\!]\gamma}} \begin{cases} C^m(\delta) & \text{if } \ell = \mathit{tt} \wedge \delta \neq \bot, \\ \{\gamma\} & \text{if } \ell = \mathit{ff}, \\ \{\bot\} & \text{otherwise.} \end{cases} \cup \bigcup_{\substack{\ell\in[\![t]\!]\gamma\\ \delta\in[\![c]\!]\gamma}} \begin{cases} \{\bot \mid \exists\gamma \in B^m(\delta)\} & \text{if } \ell = \mathit{tt} \wedge \delta \neq \bot \\ \emptyset & \text{if } \ell = \mathit{ff}, \\ \emptyset & \text{otherwise.} \end{cases}$$

$$= C^{m+1}\gamma \cup \left\{ \bot \,\middle|\, \exists\gamma \in \bigcup_{\substack{\ell\in[\![b]\!]\gamma\\ \delta\in[\![c]\!]\gamma}} \begin{cases} B^m(\delta) & \text{if } \ell = \mathit{tt} \wedge \delta \neq \bot, \\ \emptyset & \text{otherwise.} \end{cases} \right\}$$

$$= C^{m+1}\gamma \cup \{\bot \mid \exists\gamma \in B^{m+1}\gamma\}$$

□

We prove the soundness of our verification calculus by checking the soundness of each proof rule. The above lemma is used when we prove the soundness of the rule for loops.

1. (Assignment):

   Consider any state γ which validates $\exists y.\ (\!|t|\!)(y) \wedge \forall y.\ (\!|t|\!)(y) \Rightarrow \psi[y/x]$. Then, $\bot \notin [\![t]\!]\gamma$ and for any $y \in [\![t]\!]\gamma$, $\psi[y/x]$ holds.

   Now, see that $[\![x := t]\!]\gamma = \bigcup_{y\in[\![t]\!]\gamma}\{\gamma[x \mapsto y]\}$ since $\bot \notin [\![t]\!]\gamma$ and for all $y \in [\![t]\!]\gamma$, $\gamma[x \mapsto y]$ validates ψ.

2. The rule variable declarations and the rule of array assignments can be verified in a very similar manner as above and the rules of pre/postcondition strengthening/weakening, skip, and sequential compositions can be verified quite trivially.

3. (Conditional):

   Consider any state γ which validates $\phi \wedge ((\!|t|\!)(\mathit{tt}) \vee (\!|t|\!)(\mathit{ff})) \wedge \neg(\!|t|\!)(\mathit{uk})$. Then, $[\![t]\!]\gamma = \{\mathit{tt}, \mathit{ff}\}, \{\mathit{tt}\}$, or $\{\mathit{ff}\}$. Let us check the three cases:

   (a) when $[\![t]\!]\gamma = \{\mathit{tt}, \mathit{ff}\}$:

Then, γ validates $\phi \wedge (\!|t|\!)(tt)$ and $\phi \wedge (\!|t|\!)(ff)$. Therefore, (i) $\bot \notin [\![c_1]\!]\gamma$, (ii) for all $\delta \in [\![c_1]\!]\gamma$ it holds that $\delta \vDash \psi$, (iii) $\bot \notin [\![c_2]\!]\gamma$, and (iv) for all $\delta \in [\![c_2]\!]\gamma$ it holds that $\delta \vDash \psi$.

Since $\bot \notin [\![t]\!]\gamma$ and $uk \notin [\![t]\!]\gamma$, the denotation becomes $[\![\textbf{if } t \textbf{ then } c_1 \textbf{ else } c_2 \,]\!]\gamma = [\![c_1]\!]\gamma \cup [\![c_2]\!]\gamma$. Hence, the denotation does not contain $\bot$, and any resulting state δ validates ψ.

(b) when $[\![t]\!]\gamma = \{tt\}$:

Then, γ validates $\phi \wedge (\!|b|\!)(tt)$. Hence, (i) $\bot \notin [\![c_1]\!]\gamma$, (ii) for all $\delta \in [\![c_1]\!]\gamma$ it holds that $\delta \vDash \psi$. Since $[\![t]\!]\gamma = \{tt\}$, the denotation becomes $[\![\textbf{if } t \textbf{ then } c_1 \textbf{ else } c_2 \,]\!]\gamma = [\![c_1]\!]\gamma$. Therefore, $\bot$ is not in the denotation and any resulting state δ validates ψ

(c) when $[\![t]\!]\gamma = \{ff\}$, it can be done very similarly to the above item.

4. (Loop):

Consider any state γ that validates I. Then, by the side-conditions, it also validates $((\!|t|\!)(tt) \vee (\!|t|\!)(ff)) \wedge \neg(\!|t|\!)(uk)$. Hence, $[\![t]\!]\gamma = \{tt, ff\}, \{tt\}$, or $\{ff\}$ for any state γ that validates I. Now, we fix a state γ which validates I hence satisfies the precondition.

The core part of the proof is the statement: for any natural number n, it holds that (i) $\bot \notin B^n_{t,c}\gamma$, (ii) $\bot \notin C^n_{t,c}\gamma$, (iii) all δ in either $B^n_{t,c}\gamma$ or $C^n_{t,c}\gamma$ validates I, and (iv) all δ in $C^n_{t,c}\gamma$ validates $(\!|t|\!)(ff)$.

At the moment, suppose that the above statement is true. Then, all we have to show is that $B^m_{t,c}\gamma$ becomes empty as $m \in \mathbb{N}$ increases. Let us define $\ell_n \coloneqq \max\{V(\delta) \mid \delta \in B^n_{t,c}\gamma\}$ and show that ℓ_n is strictly decreasing by some quantity that is bounded below, as n increases. See that if it holds, there will be some m that for all $\delta \in B^m_{t,c}\gamma$, $[\![t]\!]\delta = \{ff\}$ and hence $B^{m+1}_{t,c}\gamma = \emptyset$.

In order to prove it, we take the two steps:

(a) If $B^1_{t,c}\gamma \neq \emptyset$, then for all $n \in \mathbb{N}$ and for all $\delta \in B^n_{t,c}\gamma$, it holds that $L(\delta) = L\gamma > 0$. In this case, let us write $\ell_0 = L\gamma$.

(b) If $B^{m+1}_{t,c}\gamma \neq \emptyset$, it holds that $\ell_{m+1} \leq \ell_m - \ell_0$.

Now, we prove each statement:

(a) $B^1_{t,c}\gamma \neq \emptyset$ only if $tt \in [\![t]\!]\gamma$ and there is some non-bottom $\delta \in [\![c]\!]\gamma$. Therefore, by the side-condition, $L\gamma > 0$.

Suppose any $\delta \in B^{m+1}_{t,c}\gamma$ for any $m \in \mathbb{N}$. See that it happens only if there is $\delta' \in B^m_{t,c}\gamma$ such that $tt \in [\![t]\!]\delta'$ and $\delta \in [\![c]\!]\delta'$. Together with Item (iii), δ' validates I and $(\!|t|\!)(tt)$. Let us define $\hat{\delta}' \coloneqq \delta' \cup (\xi \mapsto V(\delta') \cup \xi' \mapsto L(\delta')$. Since $\hat{\delta}'$ validates the precondition in the premise, we have that for any $\hat{\delta} \in [\![c]\!]\hat{\delta}'$, $\hat{\delta}$ validates I and $V \leq \xi - \xi'$ and $L = \xi'$. Hence, $L(\hat{\delta}) = L(\delta')$. Since ξ', ξ are ghost variables, $L(\hat{\delta}) = L(\delta) = L(\delta')$. In conclusion, for any $\delta \in B^{m+1}_{t,c}\gamma$, the quantity $L(\delta)$ is identical to the quantity $L(\delta')$ for some $\delta' \in B^m_{t,c}\gamma$. Since, $B^0_{t,c}\gamma = \{\gamma\}$, we conclude that they are all identical to $L\gamma$.

(b) Suppose any $\delta \in B^{m+1}_{t,c}\gamma$ for any $m \in \mathbb{N}$. See that it happens only if there is $\delta' \in B^m_{t,c}\gamma$ such that $tt \in [\![b]\!]\delta'$ and $\delta \in [\![c]\!]\delta'$. Together with Item (iii), δ' validates I and $(\!|t|\!)(tt)$. Consider $\hat{\delta}' \coloneqq \delta' \cup (\xi \mapsto V(\delta') \cup \xi' \mapsto L(\delta')$ which validates the precondition of the premise. Hence, $\delta \cup (\xi \mapsto V(\delta') \cup \xi' \mapsto L(\delta')$ validates the postcondition. Hence, $V(\delta) \leq V(\delta') - L(\delta) = V(\delta') - \ell_0$. Hence, $\ell_{m+1} \leq \ell_m - \ell_0$.

Now, we need to prove the aforementioned statement on $B^m_{t,c}$ and $C^m_{t,c}$

(a) (Base case): Recall that $B^0_{t,c}\gamma = \{\gamma\} \neq \{\bot\}$ and $C^0_{t,c}\gamma = \{\}$. Hence, the four conditions are all satisfied.

(b) (Induction step): Recall $B^{n+1}_{t,c}\gamma \coloneqq \bigcup_{\delta \in B^n_{t,c}\gamma} \bigcup_{\substack{l \in [\![b]\!]\delta \\ \delta' \in [\![c]\!]\delta}} \begin{cases} \{\delta'\} & \text{if } l = tt \wedge \delta' \neq \bot \\ \emptyset & \text{otherwise.} \end{cases}$. Since all $\delta \in B^n_{t,c}\gamma$ validates I, $[\![t]\!]\delta = \{tt\}, \{ff\}$, or $\{tt, ff\}$. In the case of $tt \in [\![t]\!]\delta$, δ validates the precondition of the premise. Hence, for all $\delta' \in [\![t]\!]\delta$, δ' is not $\bot$ and also validates I. The case of $[\![t]\!]\delta = \{ff\}$ is not of interest.

Recall $C^{n+1}_{t,c}\gamma \coloneqq C^n_{t,c}\gamma \cup \bigcup_{\delta \in B^n_{t,c}\gamma} \bigcup_{\substack{l \in [\![t]\!]\gamma \\ \delta' \in [\![c]\!]\delta}} \begin{cases} \{\delta\} & \text{if } l = ff \\ \emptyset & \text{if } l = tt \wedge \delta' \neq \bot \\ \{\bot\} & \text{otherwise.} \end{cases}$.

Since all $\gamma \in C^n_{t,c}\gamma$ validates I and $(\!|t|\!)(ff)$, we only need to care the rightmost part of the construction. Since all $\delta \in B^n_{t,c}\gamma$ validates I, by the side-condition, uk and $\bot$ are not in $[\![t]\!]\delta$. The δ is added to $C^{n+1}_{t,c}\gamma$ only if $ff \in [\![t]\!]\delta$. Therefore, δ validates both I and $(\!|t|\!)(ff)$.

Also, in the case of $tt \in [\![t]\!]\delta$, since δ validates the precondition in the premise, $\bot \notin [\![c]\!]\delta$. Therefore, $\bot \notin C^{n+1}_{t,c}$.

Issues on the Completeness

The only remaining concern for a sound formal system is completeness. Namely, if we have a correct specification $\Gamma \vdash \left[\phi\right]\ S\ \left[\psi\right] \triangleright \Gamma'$, can we derive it using our proof rules? And, this is not the case for ERC. The Hoare logic of a simple imperative language based on Presburger arithmetic is known to be incomplete. The proof first appears in [Coo78] and again together with other structures in [BT82b, BT82a].

Chapter 4. ERC in $\mathbf{Asm}(\mathbb{N}^{\mathbb{N}})$ and its Extension

In the previous chapter, the imperative programming language ERC is defined. It provides the data type R for real numbers and exact operations. Its denotational semantics is defined as set-theoretic functions. Given a well-typed program $\Gamma \vdash \mathcal{P} : \tau_1 \times \cdots \times \tau_d \to \tau$, its semantics is defined as a function from $[\![\tau_1]\!] \times \cdots \times [\![\tau_d]\!]$ to the powerdomain $\mathbb{P}([\![\tau]\!]_\perp)$. Though, it is clear from the computability of the atomic operations that was seen in Chapter 2 that the semantics is computable, we have not formalized it yet.

We want the language to be implementable in that there should be an interpreter that maps constructions of well-typed programs to type-2 machines that realize the semantics of the programs. Having an interpreter automatically asserts that our semantics is computable.

However, constructing an explicit interpreter requires making too many implementation-specific artificial decisions. For example, in order to interpret $x + y$ explicitly, we need to choose a specific realizer $F : \mathbb{N}^{\mathbb{N}} \rightharpoonup \mathbb{N}^{\mathbb{N}}$ of the real number addition function. This is meaningless that we already know the real number addition function is computable, and various realizers are already there. Hence, instead of choosing one specific realizer, we let actual developers of the language choose their favourite realizers. That means our somewhat abstract interpreter maps programs to morphisms in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$. Recall that a morphism in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ admits computable realizers but does not specify one.

In this chapter, we devise an interpreter that maps well-typed terms, commands, and programs to morphisms in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ such that the denotational semantics coincide with the definitions of the mapped morphisms. After that, we propose a rigorous way to extend ERC.

4.1 Interpretation of ERC in $\mathbf{Asm}(\mathbb{N}^{\mathbb{N}})$

We can summarize the activity of defining the denotational semantics in Chapter 3 as follows.

- For each data type τ, we defined its denotation $[\![\tau]\!] \in \mathsf{Ob}(\mathsf{Set})$ as a set.
- For each context Γ, we defined its denotation $[\![\Gamma]\!] \in \mathsf{Ob}(\mathsf{Set})$ as a set.
- For each well-typed term $\Gamma \vdash t : \tau$, we defined its denotation $[\![\Gamma \vdash t : \tau]\!] : [\![\Gamma]\!] \to \mathbb{P}([\![\tau]\!]_\perp) \in \mathsf{Mor}(\mathsf{Set})$ as a set-theoretic function.
- For each well-typed command $\Gamma \vdash c \triangleright \Gamma'$, we defined its denotation $[\![\Gamma \vdash c \triangleright \Gamma']\!] : [\![\Gamma]\!] \to \mathbb{P}([\![\Gamma']\!]_\perp) \in \mathsf{Ob}(\mathsf{Set})$ as a set-theoretic function.
- For each well-typed program $\vdash \mathcal{P} : \tau_1 \times \cdots \times \tau_d \to \tau$, we defined its denotation $[\![\vdash \mathcal{P} : \tau_1 \times \cdots \times \tau_d \to \tau]\!] : [\![\tau_1]\!] \times \cdots \times [\![\tau_d]\!] \to \mathbb{P}([\![\tau]\!]_\perp) \in \mathsf{Mor}(\mathsf{Set})$ as a set-theoretic function (though we used domain-theoretic properties).

The interpreter of ERC is a mapping $[\![\square]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$.

- For each data type τ, its interpretation $[\![\tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$ is an object in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ such that

$$\mathbf{\Gamma}([\![\tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}) = [\![\tau]\!].$$

- For each context Γ, its interpretation $[\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$ is an object in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ such that
$$\mathbf{\Gamma}([\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}) = [\![\Gamma]\!].$$
- For each well-typed term $\Gamma \vdash t : \tau$, its interpretation $[\![\Gamma \vdash t : \tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$ is a morhpism in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ such that
$$\mathbf{\Gamma}([\![\Gamma \vdash t : \tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}) = [\![\Gamma \vdash t : \tau]\!].$$
- For each well-typed command $\Gamma \vdash c \triangleright \Gamma'$, its interpretation $[\![\Gamma \vdash c \triangleright \Gamma']\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$ is a morhpism in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ such that
$$\mathbf{\Gamma}([\![\Gamma \vdash c \triangleright \Gamma']\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}) = [\![\Gamma \vdash c \triangleright \Gamma']\!].$$
- For each well-typed program $\vdash \mathcal{P} : \tau_1 \times \cdots \times \tau_d \to \tau$, its interpretation $[\![\vdash \mathcal{P} : \tau_1 \times \cdots \times \tau_d \to \tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$ is a morhpism in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ such that
$$\mathbf{\Gamma}([\![\vdash \mathcal{P} : \tau_1 \times \cdots \times \tau_d \to \tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}) = [\![\vdash \mathcal{P} : \tau_1 \times \cdots \times \tau_d \to \tau]\!].$$

In consequence, by defining the interpretation, we automatically get the fact that the denotational semantics of ERC is computable.

Given a data type, we interpret it as an assembly. The interpretation of data types are defined as follows:

$$[\![\mathsf{R}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \coloneqq \mathbf{R} \qquad [\![\mathsf{Z}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \coloneqq \mathbf{Z} \qquad [\![\mathsf{K}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \coloneqq \mathbf{K}$$

Here, $\mathbf{R}$ is any effective represented real numbers in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$, and $\mathbf{K}$ is the represented set $\flat\mathbf{2}$ from Definition 2.3 where $\flat_{\mathbf{2}} \in |\flat\mathbf{2}|$ is renamed as uk.

For a context Γ, the interpretation of it is an assembly $[\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$ of $[\![\Gamma]\!]$ where the following operations are computable for any Γ:

1. assignment $\mathsf{assign}_{x_i} : [\![x_1 : \tau_1, \cdots, x_d : \tau_d]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \times [\![\tau_i]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \to [\![x_1 : \tau_1, \cdots, x_d : \tau_d]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$ such that $\mathsf{assign}_{x_i}(\gamma, x) = \gamma[x_i \mapsto x]$,
2. extension $\mathsf{extend}_{x:\tau} : [\![x_1 : \tau_1, \cdots, x_d : \tau_d]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \times [\![\tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \to [\![x_1 : \tau_1, \cdots, x_d : \tau_d, x : \tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$ such that $\mathsf{extend}_{x:\tau}(\gamma, y) = (\gamma, (x \mapsto y))$, and
3. evaluation $\mathsf{value}_{x_i} : [\![x_1 : \tau_1, \cdots, x_d : \tau_d]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \to [\![\tau_i]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$ such that $\mathsf{value}_{x_i}\gamma = \gamma(x_i)$.

This can be done in various ways. For example, we can use a fixed enumeration on the set of variables. We omit the detail here.

4.1.1 Powerdomain in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$

Let us define an endofunctor $\mathsf{P}(\square_\perp) : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ for the powerdomain construction. That is, for any assembly $\mathbf{A}$, $\mathsf{P}(\mathbf{A}_\perp)$ is an assembly whose underlying set is $\mathbb{P}(|\mathbf{A}|_\perp)$. We let the representation relation $\Vdash_{\mathsf{P}(\mathbf{A}_\perp)}$ be induced from the injection:

$$\begin{array}{rcccl} \iota_{\mathbf{A}} & : & |\mathsf{P}(\mathbf{A}_\perp)| & \to & |\mathsf{M}(\natural\mathbf{A})| \\ & : & S & \mapsto & \bigcup_{x \in S} \begin{cases} \{\natural\} & \text{if } x = \perp, \\ \{x\} & \text{otherwise.} \end{cases} \end{array}$$

That is,

$$\varphi \Vdash_{\mathsf{P}(\mathbf{A}_\perp)} S \Leftrightarrow \varphi \Vdash_{\mathsf{M}(\natural\mathbf{A})} \iota(S) \ .$$

See that ι is simply the subset inclusion identifying $\perp$ with $\natural$. And, by how the representation relation of $\mathsf{P}(\mathbf{A}_\perp)$ is defined, $\iota_{\mathbf{A}}$ is computable by the identity function $\mathrm{id} : \mathbb{N}^{\mathbb{N}} \to \mathbb{N}^{\mathbb{N}}$.

There is a retraction of $\iota_{\mathbf{A}}$ which is the rectifying operation

$$\begin{array}{rcccl} r_{\mathbf{A}} & : & |\mathsf{M}(\natural\mathbf{A})| & \to & |\mathsf{P}(\mathbf{A}_\perp)| \\ & : & S & \mapsto & \begin{cases} S \cup \{\perp\} & \text{if } \natural \notin S \wedge S \text{ infinite}, \\ \bigcup_{x \in S} \begin{cases} \{\perp\} & \text{if } x = \perp, \\ \{x\} & \text{otherwise}, \end{cases} & \text{otherwise.} \end{cases} \end{array}$$

The function $r_{\mathbf{A}}$ renames $\natural$ to $\perp$. And, when it receives an infinite set not containing $\natural$, it adds $\perp$ in the set. See that this can be trivially done that any $\varphi \in \mathbb{N}^{\mathbb{N}}$ represents $\perp$. Hence, $r_{\mathbf{A}}$ is also computable. In consequence, they form a section-retraction pair.

The endofunctor on a morphism is defined by

$$\mathsf{P}((f : \mathbf{A} \to \mathbf{B})_\perp) = S \mapsto \bigcup_{x \in S} \begin{cases} \{f(x)\} & \text{if } x \neq \perp, \\ \{\perp\} & \text{otherwise.} \end{cases}$$

See that $\mathsf{P}((f : \mathbf{A} \to \mathbf{B})_\perp)$ is computable as it can be identified with $r_{\mathbf{B}} \circ \mathsf{M}(\natural(f)) \circ \iota_{\mathbf{A}} : \mathsf{P}(\mathbf{A}_\perp) \to \mathsf{P}(\mathbf{B}_\perp)$.

Consider two natural transformations

$$\begin{array}{rcccl} \eta_{\mathbf{A}} & : & \mathbf{A} & \to & \mathsf{P}(\mathbf{A}_\perp) \\ & : & x & \mapsto & \{x\} \end{array} \quad \text{and} \quad \begin{array}{rcccl} \mu_{\mathbf{A}} & : & \mathbf{A} & \to & \mathsf{P}(\mathbf{A}_\perp) \\ & : & S & \mapsto & \bigcup_{T \in S} \begin{cases} T & \text{if } T \neq \perp, \\ \{\perp\} & \text{otherwise.} \end{cases} \end{array}$$

The triple $(\mathsf{P}(\Box_\perp), \eta, \mu)$ is a monad that their definitions form a monad in Set. I.e., $\mathbf{\Gamma}$ is a faithful functor. Hence, the coherence conditions can be verified in Set.

Moreover, from

$$\begin{array}{rcccl} \zeta_{\mathbf{A},\mathbf{B}} & : & (\mathbf{A} \to \mathbf{B}) & \to & (\mathsf{P}(\mathbf{A}_\perp) \to \mathsf{P}(\mathbf{B}_\perp)) \\ & : & f & \mapsto & \left(S \mapsto \bigcup_{x \in S} \{f(x)\}\right) \end{array}$$

which is

$$\lambda(f : \mathbf{A} \to \mathbf{B}).\ \lambda(x : \mathsf{P}(\mathbf{A}_\perp)).\ r_{\mathbf{B}} \circ \zeta_{\mathbf{A},\mathbf{B}}^{\mathsf{M}(\natural \cdot)}\ f\ (\iota_{\mathbf{A}}\ x) \ ,$$

we can confirm that the endofunctor is a strong moand where

$$\alpha_{A,B}(S,T) = \bigcup_{x \in S \wedge y \in T} \begin{cases} \{(x,y)\} & \text{if } x \neq \perp \wedge y \neq \perp, \\ \{\perp\} & \text{otherwise,} \end{cases} \qquad \beta_{A,B}(x,S) = \bigcup_{y \in S} \begin{cases} \{(x,y)\} & \text{if } y \neq \perp \\ \{\perp\} & \text{otherwise.} \end{cases}$$

Note that the *definitions* of the endofunctor, the unit, the multiplication, tensorial strength, and α coincide with those of $\mathbb{P}(\Box_\perp) : \mathsf{Set} \to \mathsf{Set}$. That means liftings satisfy the following properties.

- When $f : \mathbf{A}_1 \times \cdots \times \mathbf{A}_d \to \mathbf{B}$, its lift $f^\dagger : \mathsf{P}((\mathbf{A}_1)_\perp) \times \cdots \times \mathsf{P}((\mathbf{A}_d)_\perp) \to \mathsf{P}(\mathbf{B}_\perp)$ by consecutively precomposing appropriate α on $\mathsf{P}(f_\perp)$ satisfies

$$f^\dagger = \mathbf{\Gamma}(f)^\dagger.$$

- When $f : \mathbf{A}_1 \times \cdots \times \mathbf{A}_d \to \mathsf{P}(\mathbf{B}_\perp)$, its lift $f^\dagger : \mathsf{P}((\mathbf{A}_1)_\perp) \times \cdots \times \mathsf{P}((\mathbf{A}_d)_\perp) \to \mathsf{P}(B_\perp)$ by consecutively precomposing appropriate α on $\mu_{\mathbf{B}} \circ \mathsf{P}(f_\perp)$ satisfies
$$f^\dagger = \mathbf{\Gamma}(f)^\dagger.$$
- When $f : \mathbf{A}_1 \times \cdots \times \mathbf{A}_d \to \mathbf{B}$, its lift it to $f^{\dagger_i} : \mathbf{A}_1 \times \mathsf{P}((\mathbf{A}_i)_\perp) \times \cdots \times \mathbf{A}_d \to \mathsf{P}(\mathbf{B}_\perp)$ by precomposing appropriate β on $\mathsf{P}(f_\perp)$ satisfies
$$f^{\dagger_i} = \mathbf{\Gamma}(f)^{\dagger_i}.$$
- When $f : \mathbf{A}_1 \times \cdots \times \mathbf{A}_d \to \mathsf{P}(\mathbf{B}_\perp)$, its lift $f^{\dagger_i} : \mathbf{A}_1 \times \mathsf{P}((\mathbf{A}_i)_\perp) \times \cdots \times \mathbf{A}_d \to \mathsf{P}(\mathbf{B}_\perp)$ by precomposing appropriate β on $\mu_{\mathbf{B}} \circ \mathsf{P}(f_\perp)$ satisfies
$$f^{\dagger_i} = \mathbf{\Gamma}(f)^{\dagger_i}.$$

Also, for a morhpism $f : \mathbf{A} \to \natural\mathbf{B}$, the codomain lifting $f^\ddagger$ which is defined by

$$f^\ddagger(x) = \begin{cases} \{f(x)\} & \text{if } f(x) \neq \natural, \\ \{\perp\} & \text{otherwise,} \end{cases}$$

is a morhpism from $\mathbf{A}$ to $\mathsf{P}(\mathbf{B}_\perp)$.

4.1.2 Interpretation of Terms, Commands, and Programs

To each well-typed term $\Gamma \vdash t : \tau$ we interpret it as a morhpism $[\![\Gamma \vdash t : \tau]\!]_{\mathsf{Asm}(\mathbb{N}^\mathbb{N})}$ from $[\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^\mathbb{N})}$ to $\mathsf{P}(([\![\tau]\!]_{\mathsf{Asm}(\mathbb{N}^\mathbb{N})})_\perp)$ such that

$$\mathbf{\Gamma}([\![\Gamma \vdash t : \tau]\!]_{\mathsf{Asm}(\mathbb{N}^\mathbb{N})}) = [\![\Gamma \vdash t : \tau]\!]$$

holds in Set.

The constants tt, ff, uk of $\mathbb{K}$ are computable with regards to $\mathbf{K}$. And, each $k \in \mathbb{Z}$ is computable with regards to $\mathbf{Z}$ and $\mathbf{R}$. The integer arithmetic $+, - : \mathbf{Z} \times \mathbf{Z} \to \mathbf{Z}$ is computable as well. And, the integer comparisons $\hat{\leq}, \hat{=}$ are computable by postcomposing the inclusion $\mathbf{2} \to \mathbf{K}$ on the ordinary integer comparisons.

Recall from Definition 2.8 that the field arithmetic operations $+, -, \times : \mathbf{R} \times \mathbf{R} \to \mathbf{R}$ are computable. And, the partial function $\square^{-1} : \mathbf{R} \rightharpoonup \mathbf{R}$ is computable[1]. That means, its $\natural$ extension $\square^{-1\natural_\natural} : \mathbf{R} \to \natural\mathbf{R}$ is computable. Since $< : \mathbf{R} \times \mathbf{R} \to \mathbf{2}$ is strongly computable, its lazy extension $<\natural_\flat$, which is $\lesssim : \mathbf{R} \times \mathbf{R} \to \mathbf{K}$ is computable.

Recall that the multivalued choice function from Example 2.12 is computable that its natural extension $\mathsf{choice}_n \natural_\natural$ is computable. See that

$$r_{\mathbf{N}} \circ \natural\mathsf{M}(\eta^\natural_{\mathbf{Z}}) \circ \iota_{\mathbf{N},\mathbf{Z}} \circ \mathsf{choice}_n \natural_\natural : \mathbf{K}^n \to \mathsf{P}(\mathbf{Z}_\perp)$$

identifying $\flat\mathbf{2}$ with $\mathbf{K}$ and where $\iota_{\mathbf{N},\mathbf{Z}} : \mathbf{N} \to \mathbf{Z}$ is the subset inclusion. See that the definition of $r_{\mathbf{N}} \circ \natural\mathsf{M}(\eta^\natural_{\mathbf{N}}) \circ \iota_{\mathbf{N},\mathbf{Z}} \circ \mathsf{choice}_n \natural_\natural$ is choose_n.

Considering the above computable functions as morphisms in $\mathsf{Asm}(\mathbb{N}^\mathbb{N})$, we can interpret well-typed terms in ERC as in Figure 4.1.

[1] we do not use the fact that it is strongly computable here.

$$
\begin{aligned}
[\![\Gamma \vdash t : \tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &: [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \to \mathsf{P}_{\perp}([\![\tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}) \\
[\![\Gamma \vdash \mathtt{true} : \mathsf{K}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &\coloneqq \lambda(\gamma : [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}).\ tt \\
[\![\Gamma \vdash \mathtt{false} : \mathsf{K}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &\coloneqq \lambda(\gamma : [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}).\ ff \\
[\![\Gamma \vdash \mathtt{undef} : \mathsf{K}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &\coloneqq \lambda(\gamma : [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}).\ uk \\
[\![\Gamma \vdash k_{\mathsf{Z}} : \mathsf{Z}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &\coloneqq \lambda(\gamma : [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}).\ k_{\mathbf{Z}} \\
[\![\Gamma \vdash k_{\mathsf{R}} : \mathsf{R}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &\coloneqq \lambda(\gamma : [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}).\ k_{\mathbf{R}} \\
[\![\Gamma \vdash x : \tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &\coloneqq \lambda(\gamma : [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}).\ \mathsf{value}^{\dagger}_{x} \\
[\![\Gamma \vdash t_1 \star t_2 : \tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &\coloneqq \lambda(\gamma : [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}).\ [\![t_1]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}\gamma \star^{\dagger} [\![t_2]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}\gamma \\
[\![\Gamma \vdash t^{-\mathbf{1}} : \mathsf{R}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &\coloneqq \lambda(\gamma : [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}).\ ([\![t]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}\gamma)^{-1\downarrow^{\ddagger\dagger}_{\natural}} \\
[\![\Gamma \vdash \mathtt{choose}_n(t_1, \cdots, t_n) : \mathsf{Z}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &\coloneqq \lambda(\gamma : [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}). \\
&\qquad (r_{\mathbf{N}} \circ \natural\mathsf{M}(\eta^{\natural}_{\mathbf{N}}) \circ \iota_{\mathbf{N},\mathbf{Z}} \circ \mathsf{choice}_n \downarrow_{\natural})^{\dagger}([\![t_1]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}\gamma, \cdots, [\![t_n]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}\gamma)
\end{aligned}
$$

Figure 4.1: The interpretation of ERC terms in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$.

Similarly, to each well-typed command $\Gamma \vdash c \triangleright \Delta$ we interpret it as a morphism $[\![\Gamma \vdash c \triangleright \Delta]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$ from $[\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$ to $\mathsf{P}([\![\Delta]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})_{\perp}})$ such that

$$\mathbf{\Gamma}([\![\Gamma \vdash c \triangleright \Delta]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}) = [\![\Gamma \vdash c \triangleright \Delta]\!]$$

in Set.

First, observe that $\mathsf{Kond}_{\mathsf{P}(\mathbf{A}_{\perp})} : \mathbf{K} \times \mathsf{P}(\mathbf{A}_{\perp}) \times \mathsf{P}(\mathbf{A}_{\perp}) \to \mathsf{P}(\mathbf{A}_{\perp})$ where $\mathsf{Kond}_{\mathsf{P}(\mathbf{A}_{\perp})}(tt, S, T) = S$, $\mathsf{Kond}_{\mathsf{P}(\mathbf{A}_{\perp})}(ff, S, T) = T$, and $\mathsf{Kond}_{\mathsf{P}(\mathbf{A}_{\perp})}(uk, S, T) = \{\perp\}$ is computable and its definition is $\mathrm{Kond}_{\mathbb{P}(|\mathbf{A}|_{\perp})}$ from Section 3.3.3.

Given $b : \mathbf{A} \to \mathsf{P}(\mathbf{K}_{\perp})$ and $c : \mathbf{A} \to \mathsf{P}(\mathbf{A}_{\perp})$, the definition of the mapping

$$\mathsf{W}_{b,c} \coloneqq \lambda(f : \mathbf{A} \to \mathsf{P}(\mathbf{A}_{\perp})).\ \mathsf{Kond}^{\dagger_1}_{\mathsf{P}(\mathbf{A}_{\perp})} \circ (b \times (f^{\dagger} \circ c) \times \eta_{\mathbf{A}})$$

coincides with

$$\mathcal{W}_{\mathbf{\Gamma}(b),\mathbf{\Gamma}(c)} : (|\mathbf{A}| \to |\mathsf{P}(\mathbf{A}_{\perp})|) \to |\mathbf{A}| \to |\mathsf{P}(\mathbf{A}_{\perp})|.$$

Therefore, by the fixed-point theorem, for each $b : \mathbf{A} \to \mathsf{P}(\mathbf{K}_{\perp})$ and $c : \mathbf{A} \to \mathsf{P}(\mathbf{A}_{\perp})$, there is the least fixed-point $f : |\mathbf{A}| \to |\mathsf{P}(\mathbf{A}_{\perp})|$ of $\mathcal{W}_{\mathbf{\Gamma}(b),\mathbf{\Gamma}(c)}$. The question is if this operation of obtaining the least fixed-point appears as a morphism in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$.

Lemma 4.1. For any assembly $\mathbf{A}$, $b : \mathbf{A} \to \mathsf{P}(\mathbf{K}_{\perp})$, and $s : \mathbf{A} \to \mathsf{P}(\mathbf{A}_{\perp})$, the fixed point of the operator $\mathsf{W}_{b,c}$ is uniformly computable. In other words, there is computable

$$
\begin{array}{rcccc}
\mathsf{WHILE}_{\mathbf{A}} & : & (\mathbf{A} \to \mathsf{P}(\mathbf{K}_{\perp})) \times (\mathbf{A} \to \mathsf{P}(\mathbf{A}_{\perp})) & \to & (\mathbf{A} \to \mathsf{P}(\mathbf{A}_{\perp})) \\
 & : & (b, c) & \mapsto & \text{the least fixed-point of } \mathcal{W}_{\mathbf{\Gamma}(b),\mathbf{\Gamma}(c)}
\end{array}
$$

Proof. Let φ_b be a name of b, φ_s be a name of c, and φ_x be a name of any $x \in |\mathbf{A}|$.

Initialize $\varphi \coloneqq \varphi_x$ and regard y as a value represented by φ.

1. While reading $\boldsymbol{\eta}_{\varphi_b}(\varphi)$, append 0 in the output tape.

2. If 1 is encountered while reading $\boldsymbol{\eta}_{\varphi_b}(\varphi)$, (that is, when $\mathit{ff} \in b(y)$ or $\bot \in b(y)$), end the procedure by appending $\varphi^{>}$ in the output tape.

3. If 2 is encountered while reading $\boldsymbol{\eta}_{\varphi_b}(\varphi)$, (that is, when $\mathit{tt} \in b(y)$ or $\bot \in b(y)$), let $\varphi \coloneqq \boldsymbol{\eta}_{\varphi_s}(\varphi)$ and repeat the procedure (1).

Verify that this procedure computes the least-fixed point. □

The well-typed commands are interpreted as in Fig. 4.2.

$$
\begin{aligned}
[\![\Gamma \vdash c \triangleright \Delta]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &: [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \to \mathsf{P}_{\bot}([\![\Delta]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}) \\
[\![\Gamma \vdash \mathbf{skip} \triangleright \Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &\coloneqq \eta_{[\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}} \\
[\![\Gamma \vdash x \coloneqq t \triangleright \Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &\coloneqq \mathsf{update}^{\dagger}_{x} \circ [\![t]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \\
[\![\Gamma \vdash \mathbf{var}\ \ x : \tau \coloneqq t\ \triangleright \Gamma']\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &\coloneqq \mathsf{extend}^{\dagger}_{x} \circ [\![t]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \\
[\![\Gamma \vdash c_1; c_2 \triangleright \Gamma']\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &\coloneqq [\![c_2]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}{}^{\dagger} \circ [\![c_1]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}\sigma \\
[\![\Gamma \vdash \mathbf{if}\ \ t\ \ \mathbf{then}\ \ c_1\ \ \mathbf{else}\ \ c_2 \triangleright \Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &\coloneqq \mathsf{Kond}^{\dagger_1}_{\mathsf{P}([\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})_{\bot}})} \circ ([\![b]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \times [\![c_1]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \times [\![c_2]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}) \\
[\![\Gamma \vdash \mathbf{while}\ \ t\ \ \mathbf{do}\ \ c \triangleright \Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &\coloneqq \mathsf{WHILE}_{\mathbf{A}}([\![t]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}, [\![c]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})})
\end{aligned}
$$

Figure 4.2: The interpretation of ERC commands in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$.

For a well-typed program

$$
\begin{aligned}
\mathcal{P} \coloneqq\ & \mathbf{function}\ (x_1 : \tau_1, \cdots, x_d : \tau_d) \\
& c \\
& \mathbf{return}\ t
\end{aligned}
$$

we interpret it as

$$
[\![\mathcal{P}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \coloneqq \lambda(v_1 : [\![\tau_1]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}).\ \cdots\ \lambda(v_d : [\![\tau_d]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}).\ [\![t]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}{}^{\dagger} \circ [\![c]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}(x_1 \mapsto v_1, \cdots, x_n \mapsto v_n)
$$

Theorem 4.1. The denotational semantics is computable in the sense that for well-typed terms t, commands c, and programs $\mathcal{P}$, it holds that $\boldsymbol{\Gamma}([\![t]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}) = [\![t]\!]$, $\boldsymbol{\Gamma}([\![c]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}) = [\![c]\!]$, and $\boldsymbol{\Gamma}([\![\mathcal{P}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}) = [\![\mathcal{P}]\!]$.

It is direct from how we defined the interpretation.

4.2 Extending ERC

4.2.1 Extension Structure

Having the categorical interpretation of ERC, it becomes clear how the language can be extended with additional data types and operations. We take the opposite way. When we introduce a data type τ, we declare its interpretation $[\![\tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \in \mathsf{Ob}(\mathsf{Asm}(\mathbb{N}^{\mathbb{N}}))$ which automatically decides the denotation $[\![\tau]\!] \coloneqq \boldsymbol{\Gamma}([\![\tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}) \in \mathsf{Ob}(\mathsf{Set})$.

A term construct is introduced as a symbol with arity $\mathtt{f} : \tau_1 \times \cdots \times \tau_d \to \tau$ where τ, τ_i are data types in the language including the newly introduced ones. The arity implies the extension of the type system with the rule

$$\frac{\Gamma \vdash t_i : \tau_i \quad (\text{for } i = 1, \cdots, d)}{\Gamma \vdash \mathtt{f}(t_1, \cdots, t_d) : \tau}$$

To the introduced term construct, we assign a morphism $\overline{\mathtt{f}} : [\![\tau_1]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \times \cdots \times [\![\tau_d]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \to \mathsf{P}([\![\tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})_\perp})$. Then, it implies the extension of the categorical interpretation by

$$[\![\Gamma \vdash \mathtt{f}(t_1, \cdots, t_d) : \tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \coloneqq \overline{\mathtt{f}}^{\dagger} \circ \big([\![t_1]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \times \cdots \times [\![t_d]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}\big).$$

And, the denotational semantics gets extended by

$$[\![\Gamma \vdash \mathtt{f}(t_1, \cdots, t_d) : \tau]\!] \coloneqq \mathbf{\Gamma}([\![f(t_1, \cdots, t_d)]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}).$$

Putting this formally, let us call $\mathcal{E} = (\mathcal{D}, \mathcal{F}, \mathcal{I})$ a *structure of ERC extension* where $\mathcal{D}$ is a set of new data types, $\mathcal{F}$ is a set of new operation symbols attached with their arities, and $\mathcal{I}$ is a mapping from $\mathcal{D}$ to $\mathsf{Ob}(\mathsf{Asm}(\mathbb{N}^{\mathbb{N}}))$ and from $\mathcal{F}$ to $\mathsf{Mor}(\mathsf{Asm}(\mathbb{N}^{\mathbb{N}}))$ that are consistently defined. That is, when $\mathtt{f} : \tau_1 \times \cdots \times \tau_d \to \tau$, $\mathcal{I}(\mathtt{f})$ has to be a morphism from $\bar{\mathcal{I}}(\tau_1) \times \cdots \times \bar{\mathcal{I}}(\tau_d)$ to $\mathsf{P}((\bar{\mathcal{I}}(\tau))_\perp)$ where $\bar{\mathcal{I}}$ is $\mathcal{I}$ extended with the assignments $\mathsf{Z} \mapsto \mathbf{Z}, \mathsf{R} \mapsto \mathbf{R}$ and $\mathsf{K} \mapsto \mathbf{K}$.

Let us see some interesting extension structures. (Recall that when there are multiple monads in the context, e.g., A, B, we write η^A to refer to the unit of A and η^B to refer to the unit of B.)

Example 4.1.

1. (ERC with Lazyness $\mathcal{E}_{\text{lazy}}$) For each data type τ, let $\mathsf{lazy}(\tau) \in \mathcal{D}$. For each data type τ, let $\mathtt{s}_\tau : \tau \to \mathsf{lazy}(\tau)$, $\mathtt{r}_\tau : \mathsf{lazy}(\tau) \to \tau \in \mathcal{F}$. Define $\mathcal{I}$ as follows.

   - $\mathcal{I}(\mathsf{lazy}(\tau)) = \flat[\![\tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$
   - $\mathcal{I}(\mathtt{s}_\tau) = \eta^{\mathsf{P}(\square_\perp)}_{\flat[\![\tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}} \circ \eta^{\flat}_{[\![\tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}}$
   - $\mathcal{I}(\mathtt{r}_\tau) = x \mapsto \begin{cases} \{x\} & \text{if } x \neq \flat\,, \\ \{\perp\} & \text{otherwise.} \end{cases}$

2. (ERC with products $\mathcal{E}_{\text{prod}}$) Let $\mathcal{D}$ be the smallest set such that $\mathsf{prod}(\tau_1, \tau_2) \in \mathcal{D}$ if $\tau_1, \tau_2 \in \mathcal{D} \cup \{\mathsf{L}, \mathsf{R}, \mathsf{Z}\}$. And, $\mathcal{F} = \{\mathtt{fst}_{\tau_1,\tau_2} : \mathsf{prod}(\tau_1, \tau_2) \to \tau_1, \mathtt{snd}_{\tau_1,\tau_2} : \mathsf{prod}(\tau_1, \tau_2) \to \tau_2, \mathtt{pair}_{\tau_1,\tau_2} : \tau_1 \times \tau_2 \to \mathsf{prod}(\tau_1, \tau_2) \mid \tau_1, \tau_2 \in \mathcal{D} \cup \{\mathsf{K}, \mathsf{Z}, \mathsf{R}\}\}$. Define $\mathcal{I}$ as follows.

   - $[\![\mathsf{prod}(\tau_1, \tau_2)]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} = [\![\tau_1]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \times [\![\tau_2]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$,
   - $[\![\mathtt{fst}_{\tau_1,\tau_2}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \coloneqq \eta^{\mathsf{P}(\square_\perp)}_{[\![\tau_1]\!]} \circ \pi_1$
   - $[\![\mathtt{snd}_{\tau_1,\tau_2}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \coloneqq \eta^{\mathsf{P}(\square_\perp)}_{[\![\tau_2]\!]} \circ \pi_2$
   - $[\![\mathtt{pair}_{\tau_1,\tau_2}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \coloneqq \eta^{\mathsf{P}(\square_\perp)}_{[\![\tau_1]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \times [\![\tau_2]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}}$

   Recall that π_1, π_2 are the projection mappings.

3. (ERC with matrices $\mathcal{E}_{\text{mat}}$) Let $\mathcal{D} = \{\mathsf{Mat}\}$, $\mathcal{F} = \{\mathsf{col} : \mathsf{Mat} \to \mathsf{Z}, \mathtt{row} : \mathsf{Mat} \to \mathsf{Z}, \mathtt{assign} : \mathsf{Mat} \times \mathsf{Z} \times \mathsf{Z} \times \mathsf{R} \to \mathsf{Mat}, \mathtt{value} : \mathsf{Mat} \times \mathsf{Z} \times \mathsf{Z} \to \mathsf{R}, \mathtt{E} : \mathsf{Z} \times \mathsf{Z} \to \mathsf{Mat}\}$. Define $\mathcal{I}$ as follows.

- $\mathcal{I}(\mathsf{Mat})$ is the assembly $\mathbf{Mat}(\mathbf{R})$ of the set of real matrices whose representation relation is induced from the injection

$$X \in \mathbb{R}^{n\times m} \mapsto (n, m, X_{1,1}, X_{1,2}, \cdots, X_{n,m}, 0, 0 \cdots) \in |\mathbf{Z} \times \mathbf{Z} \times (\mathbf{N} \to \mathbf{R})|.$$

See that the following functions are computable:

$$\begin{array}{rcccl}
\overline{\texttt{col}} & : & \mathbf{Mat}(\mathbf{R}) & \to & \mathsf{P}(\mathbf{Z}_\perp) \\
& : & X & \mapsto & \{\text{the number of columns in } X\} \\
\overline{\texttt{row}} & : & \mathbf{Mat}(\mathbf{R}) & \to & \mathsf{P}(\mathbf{Z}_\perp) \\
& : & X & \mapsto & \{\text{the number of rows in } X\} \\
\overline{\texttt{assign}} & : & \mathbf{Mat}(\mathbf{R}) \times \mathbf{Z} \times \mathbf{Z} \times \mathbf{R} & \to & \mathsf{P}(\mathbf{Z}_\perp) \\
& : & (X, i, j, x) & \mapsto & \begin{cases} \{\text{copy of } X \text{ with } X_{i,j} = x\} & \text{if } X \in \mathbb{R}^{n\times m} \\ & \wedge 1 \le i, j \le n, \\ \{\perp\} & \text{otherwise.} \end{cases} \\
\overline{\texttt{value}} & : & \mathbf{Mat}(\mathbf{R}) \times \mathbf{Z} \times \mathbf{Z} & \to & \mathsf{P}(\mathbf{R}_\perp) \\
& : & (X, i, j) & \mapsto & \begin{cases} \{X_{i,j}\} & \text{if } X \in \mathbb{R}^{n\times m} \wedge 1 \le i \le n \wedge 1 \le j \le m, \\ \{\perp\} & \text{otherwise.} \end{cases} \\
\overline{\texttt{E}} & : & \mathbf{Z} \times \mathbf{Z} & \to & \mathsf{P}(\mathbf{Mat}_\perp) \\
& : & (i, j) & \mapsto & \begin{cases} \{\text{the } i \times j \text{ identity matrix}\} & \text{if } 0 < i, j, \\ \{\perp\} & \text{otherwise.} \end{cases}
\end{array}$$

For each $\texttt{f} \in \mathcal{F}$, define $\mathcal{I}(\texttt{f}) = \overline{\texttt{f}}$.

4. (ERC with continuous real functions $\mathcal{E}_{\text{rfun}}$) Let $\mathcal{D} = \{\mathsf{C}(\mathsf{R}, \mathsf{R})\}$. And, $\mathcal{F} = \{\texttt{eval} : \mathsf{C}(\mathsf{R}, \mathsf{R}) \times \mathsf{R} \to \mathsf{R}\}$. Define $\mathcal{I}$ as follows.
   - $\mathcal{I}(\mathsf{C}(\mathsf{R}, \mathsf{R})) \coloneqq (\mathbf{R} \to \mathbf{R})$ and
   - $\mathcal{I}(\texttt{eval}) \coloneqq \eta_{\mathbf{R}}^{\mathsf{P}(\square_\perp)} \circ \mathsf{eval} : (\mathbf{R} \to \mathbf{R}) \times \mathbf{R} \to \mathsf{P}(\mathbf{R}_\perp)$ where $\mathsf{eval} : (\mathbf{R} \to \mathbf{R}) \times \mathbf{R} \to \mathbf{R}$ is the evaluation map in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$.

4.2.2 Extended Reasoning Principles

Note that our extended Hoare logic is for commands, and the term language has been dealt with by the translation function $(\!|\square|\!)$. Hence, when we extend our specification language thus that it makes the extended term-language definable, we still have a sound verification calculus.

Definition 4.1.

1. A ERC extension structure is *single-valued total* if for each added operation $\texttt{f} : \tau_1 \times \cdots \times \tau_d \to \tau$, its interpretation is given by $\mathcal{I}(\texttt{f}) = \eta^{\mathsf{P}(\square_\perp)}_{[\![\tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}} \circ \hat{f}$ for some $\hat{f} : [\![\tau_1]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \times \cdots \times [\![\tau_d]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \to [\![\tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$. See that $\mathcal{E}_{\text{lazy}}$ and $\mathcal{E}_{\text{mat}}$ are not single-valued total but $\mathcal{E}_{\text{prod}}$ and $\mathcal{E}_{\text{rfun}}$ are single-valued total.

2. Given a single-valued total extension $\mathcal{E} = (\mathcal{D}, \mathcal{F}, \mathcal{I})$, define a structure $\mathcal{S}(\mathcal{E})$ by extending $\mathcal{S}$ with sorts $\mathbf{\Gamma}([\![\tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})})$ for each $\tau \in \mathcal{D}$ and functions $\mathbf{\Gamma}([\![\hat{f}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})})$ for each $\texttt{f} \in \mathcal{F}$. Let $\mathcal{T}(\mathcal{E})$ be the first order theory over the extended structure and $\mathcal{L}(\mathcal{E})$ be the first order language over the extended structure.

Then, the translation function $(\!|-|\!)$ can easily get extended to the extended-term language:

$$(\!|\mathtt{f}(t_1, \cdots, t_d)|\!)(x) = \exists x_1, \cdots, x_d.\ (\!|t_1|\!)(x_1) \wedge \cdots \wedge (\!|t_d|\!)(x_d) \wedge x = \hat{f}(x_1, \cdots, x_d)$$

Remark 4.1. The decidability property of course does not get preserved. For example, $\mathcal{E} = (\{\}, \{\times : \mathsf{Z} \times \mathsf{Z} \to \mathsf{Z}\}, \mathcal{I})$ where $\mathcal{I}(\times)$ is the integer multiplication function makes the extended logical language expressive for Peano arithmetic.

Definition 4.2. Given a single-valued total extension $\mathcal{E}$, the verification calculus of ERC($\mathcal{E}$) is the verification calculus of ERC with $(\!|\Box|\!)$ extended.

4.3 Root Finding in ERC Extended with Continuous Real Functions

The problem of finding a root to a real function f occurs frequently in numerical practices. Here, we provide a program for the problem in the case where f is continuous and admits a single root in a given interval (a, b) with a promise that $f(a) < 0 < f(1)$; i.e., we consider an algorithmic version of the Intermediate-Value Theorem. This case is commonly treated using Bisection method. By testing the sign of $f(x)$ where x is the mid point $x = (a + b)/2$, refine the interval to (a, x) or (x, b) accordingly. However, this method fails in the case when $f(x)$ is exactly the root of f. Instead, *Trisection* [Her96, p. 336] tests the signs of $f(x)$ and $f(y)$ in parallel where $x = (2a + b)/3$ and $y = (a + 2b)/3$. With the promise that f admits a single root in (a, b), at least one of the two parallel tests succeeds. Hence, we can safely refine the interval to either (a, y) or (x, b). Repeating this refinement until the width of the interval gets small than 2^{-p}, we get a 2^{-p} approximation to the root of f. See Fig. 4.3 where $f(t)$ is an abbreviation for (f, t) for any term t.

```
trisection ≔ function (f : C(R, R), p : Z)
               var a : R ≔ 0;
               var b : R ≔ 1;
               while choose₂(b − a ≲ 2^p, 2^{p−1} ≲ b − a) ≙ 2 do
                 if choose₂(f(2×a/3 + b/3)×f(b) ≲ 0, f(a) × f(a/3 + 2×b/3) ≲ 0) ≙ 1 then
                   a ≔ 2×a/3 + b/3
                 else
                   b ≔ a/3 + 2×b/3
               return a
```

Figure 4.3: A root finding program in ERC($\mathcal{E}_{\mathrm{rfun}}$).

See that the loop continues until $b - a < 2^p$ and it is promised that the loop ends when $b - a \leq 2^{p-1}$. In each iteration, it tests if $f(x) \times f(b) < 0$ or $f(a) \times f(y) < 0$ in parallel where x is the one-third point $(2a + b)/3$ and y is the two-third point $(a + 2b)/3$. And, it refines the interval accordingly. Therefore, when the loop exits, one endpoint, a, is 2^p approximation to the root of f.

Let us prove the correctness of trisection to illustrate formal verification in ERC. To emphasize, our purpose here is not to actually establish correctness of the long-known Trisection method, but to

demonstrate our proof rules using a toy example. Since Trisection relies on the Intermediate Value Theorem, any correctness proof must make full use of real (as opposed to, say, floating-point, rational, or algebraic) numbers.

Let us define some abbreviations such that the algorithm in Figure 4.3, for any continuous real function f having a simple root in $(0,1)$ becomes of the form **function** $(f : \mathsf{C}(\mathsf{R},\mathsf{R}), p : \mathsf{Z})\ c_1; c_2$ **return** a.

$$
\begin{array}{lcl}
\tilde{t}_1 & \coloneqq & b - a \lesssim 2^p\ ,\ \ \tilde{t}_2 \coloneqq\ 2^{p-1} \lesssim b - a \\
t_1 & \coloneqq & f(2\times a/3 + b/3)\times f(b) \lesssim 0\ ,\ \ t_2 \coloneqq f(a)\times f(a/3 + 2\times b/3) \lesssim 0 \\
b_1 & \coloneqq & \mathtt{choose}_2(\tilde{t}_1, \tilde{t}_2) \mathrel{\hat{=}} 2,\ \ b_2 \coloneqq\ \mathtt{choose}_2(t_1, t_2) \mathrel{\hat{=}} 1 \\
c_1 & \coloneqq & \textbf{var}\ a : \mathsf{R} \coloneqq 0; \textbf{var}\ b : \mathsf{R} \coloneqq 1 \\
c_2 & \coloneqq & \textbf{while}\ b_1\ \textbf{do}\ c_3 \\
c_3 & \coloneqq & \textbf{if}\ b_2\ \textbf{then}\ c_4\ \textbf{else}\ c_5 \\
c_4 & \coloneqq & a \coloneqq 2\times a/3 + b/3 \\
c_5 & \coloneqq & b \coloneqq a/3 + 2\times b/3
\end{array}
$$

We want the program to realize a real functional that computes a root of f, provided that f has a unique root in $(0,1)$ and that the signs of $f(0)$, $f(1)$ are different. In order to verify that the program meets the desired property, we need to show that under the condition, the following hold: (i) $\bot \notin [\![c_1; c_2]\!]\sigma$ and (ii) for all resulting states $\delta \in [\![c_1; c_2]\!]\sigma$, $\delta(a)$ is a $2^{p'}$ approximation of the unique root of f, for any $p' \in \mathbb{Z}$ where $\sigma(p) = p'$.

The specification language we use is the logic of ERC extended with $\mathcal{E}_{\mathrm{rfun}}$. That is, in the language it has the additional sort $\mathcal{C}(\mathbb{R},\mathbb{R})$ which is the set of continuous real functions. It contains the evaluation map $\mathsf{eval} : \mathcal{C}(\mathbb{R},\mathbb{R}) \times \mathbb{R} \to \mathbb{R}$. As we did for the programming language, let us abbreviate $\mathsf{eval}(f, x)$ by $f(x)$. Let $\mathsf{uniq}(f, x, y)$ be the predicate

$$\mathsf{uniq}(f \in \mathcal{C}(\mathbb{R},\mathbb{R}), x \in \mathbb{R}, y \in \mathbb{R}) \equiv f(x) \times f(y) < 0 \wedge \exists! z \in (x, y).\ f(z) = 0.$$

Here, $\exists! x.\ P(x)$ is an abbreviation for $\exists x.\ P(x) \wedge \forall y\ z.\ P(x) \wedge P(y) \Rightarrow x = y$.

The specification we wish to have is as follows:

$$\Gamma \vdash \big[p = p' \wedge \mathsf{uniq}(f, 0, 1)\big] c_1; c_2 \big[\exists! z.\ f(z) = 0 \wedge 0 < z < 1 \wedge |a - z| \le 2^{p'}\big] \triangleright \Gamma'$$

where $\Gamma = p, p' : \mathsf{Z}$, $\Gamma' = p, p' : \mathsf{Z}, a, b, \epsilon : \mathsf{R}$. Here, p' is an auxiliary variable that stores the initial value of p considering that the value p stores may vary (though it does not in this specific example) at the end state. The post condition says, when $c_1; c_2$ terminates, the return value a is a $2^{p'}$ approximation of the *unique* root of f. Hence, the specification ensures that the program that computes the root.

Implicitly replacing $\lesssim$ with $<$, the terms $t_1, t_2, \tilde{t}_1, \tilde{t}_2$ can be interpreted as formulae in our specification language. See that $(\!|b_1|\!)(tt) \Leftrightarrow \tilde{t}_2$, $(\!|b_1|\!)(ff) \Leftrightarrow \tilde{t}_1$, $\neg(\!|b_1|\!)(uk) \Leftrightarrow \top$, $(\!|b_2|\!)(tt) \Leftrightarrow t_1$, $(\!|b_2|\!)(ff) \Leftrightarrow t_2$, and $\neg(\!|b_2|\!)(uk) = \top$ hold.

Let us define $I \coloneqq p = p' \wedge 0 \le a < b \le 1 \wedge \mathsf{uniq}(f, a, b) \wedge \mathsf{uniq}(f, 0, 1)$ as a candidate for the loop invariant, $V \coloneqq b - a - 2^{p-1}$ as a candidate for the loop variant, $L \coloneqq 2^{p-2}$ be a candidate for a lower bound decrement, $\tilde{P} \coloneqq \tilde{t}_2 \wedge I \wedge V = \xi \wedge L = \xi'$, and $\tilde{Q} \coloneqq I \wedge V \le \xi - \xi' \wedge L = \xi'$ in our specification language with variables ξ, ξ' of the sort $\mathbb{R}$. Let $\Delta \coloneqq p, p' : \mathsf{Z}, a, b, \xi, \xi' : \mathsf{R}$.

From the axiom for assignments, we have the triples:

$$\Delta \vdash \big[\exists \omega.\ (\!|2 \times a/3 + b/3|\!)(\omega) \wedge \forall \omega.\ (\!|2 \times a/3 + .b/3|\!)(\omega) \Rightarrow \tilde{Q}[\omega/a]\big]\ c_4\ \big[\tilde{Q}\big] \triangleright \Delta$$

$$\Delta \vdash \left[\exists \omega.\ (\!|a/3 + 2 \times b/3|\!)(\omega) \wedge \forall \omega.\ (\!|a/3 + 2 \times b/3|\!)(\omega) \Rightarrow \tilde{Q}[\omega/b]\right]\ c_5\ \left[\tilde{Q}\right] \triangleright \Delta$$

See that we can apply the rule of precondition weakening to get the following triples derived:

$$\Delta \vdash \left[\tilde{Q}[(2 \times a/3 + b/3)/a]\right]\ c_4\ \left[\tilde{Q}\right] \triangleright \Delta, \quad \Delta \vdash \left[\tilde{Q}[(a/3 + 2 \times b/3)/b]\right]\ c_5\ \left[\tilde{Q}\right] \triangleright \Delta.$$

When we unwrap the abbreviations, we have

$$\begin{aligned}
&\tilde{Q}[(2 \times a/3 + b/3)/a] \equiv \\
&\quad p = p' \wedge 0 \leq (2 \times a/3 + b/3) < b \leq 1 \wedge \mathsf{uniq}(f, (2 \times a/3 + b/3), b) \wedge \mathsf{uniq}(f, 0, 1) \\
&\wedge \quad b - (2 \times a/3 + b/3) - 2^{p-1} \leq \xi - \xi' \\
&\wedge \quad 2^{p-2} = \xi'
\end{aligned}$$

and

$$\begin{aligned}
&\tilde{P} \wedge t_1 \equiv \\
&\quad 2^{p-1} < b - a. \\
&\wedge \quad p = p' \wedge 0 \leq a < b \leq 1 \wedge \mathsf{uniq}(f, a, b) \wedge \mathsf{uniq}(f, 0, 1) \\
&\wedge \quad b - a - 2^{p-1} = \xi \\
&\wedge \quad 2^{p-2} = \xi' \\
&\wedge \quad f(2 \times a/3 + b/3) \times f(b) < 0
\end{aligned}$$

See that $\tilde{P} \wedge t_1 \Rightarrow \tilde{Q}[(2 \times a/3 + b/3)/a]$ holds using intermediate value theorem that if an interval (a, b) contains a root of f uniquely, and if $f(x) \times f(y) < 0$ for $a \leq x < y \leq b$, then (x, y) also contains the root of f uniquely. And, similarly, $\tilde{P} \wedge t_2 \Rightarrow \tilde{Q}[(a/3 + 2 \times b/3)/b]$ holds.

After having the implications proven, we can use the rule of precondition strengthening on the triples of c_4, c_5, and apply the rule for conditionals to get the triple:

$$\Delta \vdash \left[\tilde{t}_2 \wedge I \wedge \left(V = \xi\right) \wedge L = \xi'\right]\ c_3\ \left[I \wedge (V \leq \xi - \xi') \wedge L = \xi'\right] \triangleright \Delta$$

The side-conditions of the rule for while loops are quite trivial. Hence, assuming that they are proven, we apply the rule of while loops, apply the rules of assignments and sequential compositions, and we get the following triple:

$$\Gamma \vdash \left[I[0/a, 1/b]\right]\ c_1; c_2\ \left[I \wedge \tilde{t}_2\right] \triangleright \Gamma'$$

Using the rule of pre/postcondition strengthening/weakening, we can get the originally desired specification.

Chapter 5. Clerical: Expression-based Language with Limit Operator

Our work began as an attempt to fill a lacuna in Chapter 3. Our goal is to make an imperative programming language that supports the functionality of constructing real numbers via limit operations while it remains simple in the sense that it does not introduce functions as first-class citizens.

Refraining from introducing function types in the language can be achieved by interpreting an *expression* with a free integer variable as a sequence. Our limit operation is of the form $\texttt{lim}(n, e)$. Here, e is a *real* typed expression containing a free integer variable n. If the induced function $n \mapsto e$ defines a rapid Cauchy sequence, the limit expression defines the real number that the sequence converges to.

In order to be able to define a useful class of limits in the above way, the expression-language must be rich enough for e to be able to define interesting functions $n \mapsto e$. To achieve this, it is inevitable to make expressions *command-like*, in that they are allowed to contain loops performing much-complicated computation. In general, such command-like expressions, which subsume *commands*, may have side effects. Nevertheless, we distinguish between uses of expressions in which side-effects are allowed (e.g., when they are used as commands) and uses in which *purity* (i.e., side-effect-free) is required. For example, the expression e in $\texttt{lim}(n{:}\mathsf{Z}, e)$ is required to be pure, as the value of the expression e, on different values of n, must not depend on the strategy for evaluating such approximating expressions, which is considered implementation-specific.

In this chapter, we propose *Clerical* (Command-Like Expressions for Real Infinite-precision Calculations) as a streamlined imperative language for real number computation that combines real-valued variables with a limit operation.

Having a limit operation explicitly, our denotational semantics has to care about limits of non-converging sequence. Basically, Clerical provides five different operations that cause partialities: i) comparing real numbers, ii) division by 0, iii) infinite loop, and iv) non-converging limit. Recall from Chapter 2 that the partial functions (i) and (ii) are strongly computable that their $\flat$ extensions are computable. Also, the partiality caused infinite loop is by definition $\flat$ kind of partiality. On the other hand, the partiality of the limit operator is totally different in that it is a weakly computable partial function. That is, its $\sharp$ extension is computable. Hence, in our denotational semantics, we strictly distinguish the two partialities. We denote $\bot$ for $\flat$, thus that it indicates nontermination. Meanwhile, we introduce a new symbol $\texttt{e}$ for $\natural$ that indicates general failure. (As $\sharp \to \natural$ and $\flat \to \natural$ hold.) Hence, we make a limit operation's denotation to be $\texttt{e}$ when it receives an invalid sequence; and the denotations of comparing the same real number, dividing by zero, and infinite loop are $\bot$. [1]

5.1 Overview of Clerical with Example Programs

Before going through the formal construction of the language in Section 5.2, let us see how programs in Clerical look like, intuitively, by seeing through some examples.

Clerical is an expression-based imperative programming language where expressions subsume commands. For example, variable assignments ($x := e$), loops ($\texttt{while}\ e\ \texttt{do}\ c\ \texttt{end}$), and conditional statements

[1] Note that in ERC, we only distinguished comparing the same real number by *uk*, and let all partialities be indicated by $\bot$. This is the core difference between the denotational semantics of the two languages.

(`if` e `then` c_1 `else` c_2 `end`) all belong to the set of expressions. In Clerical, expressions, in general, do not only represent values but also modify states.

However, we do not want every expression to be side-effecting. For example, when we have an expression of the form $e_1 + e_2$, it makes the language overly complicated when it is possible that the evaluation of e_1 and the evaluation of e_2 both modify the same variable. In this case, we have to specify the protocol on reading and writing to shared memory.

We call an expression *pure* if it is side-effect-free. We enforce expressions that ought to purely represent values pure. For example, in an arithmetical expression $e_1 \odot e_2$, the sub-expressions e_1 and e_2 have to be pure, where $\odot$ is one of the operators $+, -, \times$ of the integer arithmetic or $+, -, \times$ of the real arithmetic. (Hence, any arithmetical expression is also pure.) In consequence, we do not need to specify in which order the sub-expressions evaluate; the evaluations of e_1 and e_2 are independent. Similarly, the expressions in assignments and the conditions in loops and conditionals must be pure as well.

Alongside enforcing some expressions side-effect-free, we still want those pure expressions to be expressive. We allow expressions to create their own mutable local variables. As long as an expression assigns values only to its local variables, it is not side-effecting. By allowing it, a pure expression still can be expressive. As an example, the below expression named pos_prec is pure since it only assigns to its local variables x and m:

$$\begin{aligned}\mathsf{pos\_prec}(n : \mathsf{Z}) \coloneqq\ & \texttt{var}\ x \coloneqq \iota(1)\ \texttt{in}\\ & \texttt{var}\ m \coloneqq n\ \texttt{in}\\ & \texttt{while}\ m > 0\ \texttt{do}\\ & \quad x \coloneqq x \times \iota(2);\\ & \quad m \coloneqq m - 1\\ & \texttt{end}; x\end{aligned}$$

Here, $0, 1, 2$ are integer constants. And, ι is the coercion operations such that $\iota(2)$ represents the real number $2 \in \mathbb{R}$. When e_1 and e_2 are expressions, $e_1; e_2$ is the expression that represents the sequential composition of e_1 and e_2: execute e_1 then execute e_2. The value of $e_1; e_2$ is the value of e_2. The parameter $n : \mathsf{Z}$ refers to an arbitrary expression of type Z for integers. The intended meaning of $\mathsf{pos\_prec}(n : \mathsf{Z})$ is 2^n for a non-negative integer n.

We can write an expression that works on all integers as follows:

$$\mathsf{prec}(k : \mathsf{Z}) \coloneqq \quad \texttt{if}\ k > 0\ \texttt{then}\ \mathsf{pos\_prec}(k)\ \texttt{else}\ \iota(1)/\mathsf{pos\_prec}(-k)\ \texttt{end}$$

Though $\mathsf{pos\_prec}(n)$ is written as a function call, it is simply an abbreviation. The expression $\mathsf{pos\_prec}(n)$ means to copy the definition of pos_prec with the free occurrences of the variable n substituted by m. Note that this is not a functionality of Clerical.

Limit operations in Clerical are given by a construct of the form `lim` $n\,.\,e$ where e is a *pure* expression of type R for real numbers, that contains a free integer variable n. When the value of e forms a rapid Cauchy sequence as n grows to ∞, the value of the limit expression is defined to be the real number that the sequence converges to. As an example, `lim` $n\,.\,\mathsf{prec}(-n)$ evaluates to 0.

Of course, we can define more complicated limits:

$$\begin{aligned}\mathsf{partial\_sqrt}(x : \mathsf{R}) \coloneqq \mathtt{lim}\ n.\ &\mathtt{var}\ a \coloneqq \iota(0)\ \mathtt{in}\\ &\mathtt{var}\ b \coloneqq x + \iota(1)\ \mathtt{in}\\ &\mathtt{while}\ \mathsf{prec}(-n) \hat{>} b - a\ \mathtt{do}\\ &\quad \mathtt{var}\ m \coloneqq (b+a)/\iota(2)\ \mathtt{in}\\ &\quad \mathtt{if}\ m \times m - x \hat{>} \iota(0)\ \mathtt{then}\ b \coloneqq m\ \mathtt{else}\ a \coloneqq m\ \mathtt{end}\\ &\mathtt{end}; a\end{aligned}$$

The expression inside of the limit in $\mathsf{partial\_sqrt}$ performs Bisection method to find the root of $f(y) = y^2 - x$ in the interval $[0, 1 + x]$. Throughout the iterations, the interval $[a, b]$ gets refined until the width of the interval gets less than 2^{-n}. Hence, when the loop is escaped, one endpoint of the interval, a, is 2^{-n} approximation of $\sqrt{x}$. Therefore, for a real number x which ensures the loop to be escaped for every positive integer n, the expression $\mathsf{partial\_sqrt}(x)$ evaluates to the positive square root of x.

However, the expression $\mathsf{partial\_sqrt}$ is not totally defined. As it is necessary for real number computation, comparison tests are defined partially also in Clerical; i.e., testing $e_1 \hat{>} e_2$, which is of type B for Booleans, where e_1, e_2 are expressions of type R, does not terminate when they represent the same real number. Depending on the values that n and x represent, the expression in the limit diverges; i.e., for certain x, the sequence that the limit gets provided is not a proper Cauchy sequence. Hence, for such x, the limit is not well-defined. (For example, when $x = 1$, for any positive n, at the first iteration, $m = 1$. Hence, the evaluation of the condition $0 \hat{>} 0$ will never terminate.)

Nondeterminism in Clerical, which is essential to make partial comparison tests useful, is provided by Dijkstra-style guarded commands:

$$\begin{aligned}&\mathtt{case}\\ &\mid b_1 \Rightarrow c_1\\ &\mid b_2 \Rightarrow c_2\\ &\mathtt{end}\end{aligned}$$

where b_1, b_2 are pure Boolean expressions and c_1, c_2 are (possibly side-effecting) expressions. The intended meaning is that c_1 may execute if e_1 is true, and c_2 may execute if b_2 is true. When b_1 and b_2 both hold, either branch may execute nondeterministically. Even when one of the evaluations of the guards does not terminate, if the other holds, the corresponding branch executes. For example, we can write a program for an approximate test $x > y$ with precision 2^{-n} by $\mathtt{case}\ x > y + 2^{-n} \Rightarrow \mathtt{true} \mid y > x + 2^{-n} \Rightarrow \mathtt{false}\ \mathtt{end}$.

See that the following expression correctly computes the maximum of two real expressions x, y:

$$\begin{aligned}\mathsf{max}(x\ y : \mathsf{R}) \coloneqq \mathtt{lim}\ n.\ &\mathtt{case}\\ &\mid x \hat{>} y - \mathsf{prec}(-n) \Rightarrow x\\ &\mid y \hat{>} x - \mathsf{prec}(-n) \Rightarrow y\\ &\mathtt{end}\end{aligned}$$

When $x \geq y$, even if the second branch is taken as $y > x - 2^{-n}$, since $|x - y| < 2^{-n}$, we can ensure that y approximates the maximum, which is x, by 2^{-n}. Using max, we can also define expressions for computing $\mathsf{min}(x\ y : \mathsf{R}) \coloneqq -\mathsf{max}(-x, -y)$ and $\mathsf{abs}(x : \mathsf{R}) \coloneqq \mathsf{max}(x, -x)$.

Now, back to the problem of finding the root of a quadratic polynomial. We know already from Section 4.3 that Trisection replaces Bisection. The following expression correctly computes the positive

square root of x:

$$
\begin{array}{l}
\mathsf{sqrt}(x : \mathsf{R}) \coloneqq \mathtt{lim}\ n.\ \mathtt{var}\ a \coloneqq \iota(0)\ \mathtt{in} \\
\qquad\qquad \mathtt{var}\ b \coloneqq x + \iota(1)\ \mathtt{in} \\
\qquad\qquad \mathtt{while\ case}\ b - a \mathbin{\hat{>}} \mathsf{prec}(-n-1) \Rightarrow \mathtt{true} \mid \mathsf{prec}(-n) \mathbin{\hat{>}} b - a \Rightarrow \mathtt{false\ end\ do} \\
\qquad\qquad\quad \mathtt{var}\ c \coloneqq (\iota(2) \times a + b)/\iota(3)\ \mathtt{in} \\
\qquad\qquad\quad \mathtt{var}\ d \coloneqq (a + \iota(2) \times b)/\iota(3)\ \mathtt{in} \\
\qquad\qquad\quad \mathtt{case} \\
\qquad\qquad\quad \mid \iota(0) \mathbin{\hat{>}} c \times c - x \Rightarrow a \coloneqq c \\
\qquad\qquad\quad \mid \iota(0) \mathbin{\hat{>}} d \times d - x \Rightarrow b \coloneqq d \\
\qquad\qquad\quad \mathtt{end} \\
\qquad\qquad \mathtt{end}; a
\end{array}
$$

5.2 Formal Syntax and Typing

In this section, we introduce the grammar of Clerical including the typing rules.

5.2.1 Formal Syntax

Clerical provides the following data types: Z for the set of integers, B for the set of Booleans, R for the set of real numbers, and U for the singleton set $\{*\}$. We write τ, σ and their variants to refer to arbitrary data types. As usual, we assume there is unlimited supplies of variables and write them using alphabets $x, y, v, \cdots$ and their variants.

Clerical is an expression-based language where an expression stands for both a computational instruction and a value. We write small alphabets e, c and their variants to refer to arbitrary expressions. Although they are all just expressions, we make some (typing-level) distinctions: an expression is pure if the expression only assigns to its local variables; i.e., an expression is pure if it is side-effect-free. Though the purity of an expression is dealt with in typing rules, in the definition of expressions, we write e and its variants to denote expressions that ought to be pure, and c and its variants to denote expressions that are possibly side-effecting. See Fig. 5.1 for the definition.

We abbreviate e_1/e_2 for $e_1 \times {e_2}^{-1}$, $-e$ for $0 - e$, $-e$ for $\iota(0) - e$, $e_1 > e_2$ for $e_2 < e_1$, and $e_1 \mathbin{\hat{>}} e_2$ for $e_2 \mathbin{\hat{<}} e_1$.

5.2.2 Typing Rules

Not all expressions are of interest; i.e., not every expression has a meaning. We are interested only in well-typed expressions and an expression being well-typed is dependent on *contexts* which is a structure that memorizes in which data type that a variable has been declared.

A *typing context* is a function from a finite set of variables to their data types where $\cdot$ denotes the empty function. For a typing context Γ, a variable x not in $\mathrm{dom}(\Gamma)$, and a data type τ, we write $\Gamma, x{:}\tau$ to denote the function Γ extended with the mapping $x \mapsto \tau$. For typing contexts Γ and Δ, when their domains are disjoint, we write Γ, Δ to denote the join of the two functions. A typing context being a function from a finite set, we often write it as a list of assignments: $x_1 : \tau_1 \cdots x_n : \tau_n$.

A *read-only* context Γ is a typing context. And, a *read-write* context is a pair of typing contexts $\Gamma; \Delta$ where Γ is a typing context of read-only variables and Δ is a typing context of read-write variables. Of course, the domains of Γ and Δ have to be disjoint.

Expression e, c	$::= x$	variable
	$\mid$ `true` $\mid$ `false`	boolean constant
	$\mid k$	integer constant
	$\mid$ `skip`	unit
	$\mid \iota(e)$	coercion from Z to R
	$\mid e_1 \odot e_2$	integer arithmetic $\odot \in \{+, -, \times\}$
	$\mid e_1 \boxdot e_2 \mid e^{-1}$	real arithmetic $\boxdot \in \{+, -, \times\}$
	$\mid e_1 < e_2 \mid e_1 = e_2$	integer comparison
	$\mid e_1 \hat{<} e_2$	real comparison
	$\mid$ `lim` x `.` e	limit (x bound in e)
	$\mid c_1; c_2$	sequencing
	$\mid$ `var` $x := e$ `in` c	local variable (x bound in c)
	$\mid x := e$	assignment
	$\mid$ `if` e `then` c_1 `else` c_2 `end`	conditional
	$\mid$ `case` $e_1 \Rightarrow c_1 \mid \cdots \mid e_n \Rightarrow c_n$ `end`	guarded cases
	$\mid$ `while` e `do` c `end`	loop

Figure 5.1: The formal syntax of Clerical expressions.

Having two different types of contexts, we define two different judgement forms accordingly:

$$\Gamma \vdash e : \tau \qquad e \text{ has type } \tau \text{ in read-only context } \Gamma$$
$$\Gamma; \Delta \Vdash c : \tau \qquad c \text{ has type } \tau \text{ in read-write context } \Gamma; \Delta$$

The typing rules are defined in Fig. 5.2.

See that as we demand e to be pure and c be not in $\texttt{var}\ x := e\ \texttt{in}\ c$, the rule for deriving $\Gamma; \Delta \Vdash \texttt{var}\ x := e\ \texttt{in}\ c : \tau$ requires $\Gamma, \Delta \vdash e : \sigma$ and $\Gamma; \Delta, x : \sigma \Vdash c : \tau$. In words, e has to be a pure expression of type σ and c has to be a read-write expression of type τ assuming $x : \sigma$.

There are two notable typing rules:

$$\frac{\Gamma; \cdot \Vdash e : \tau}{\Gamma \vdash e : \tau} \qquad\qquad \frac{\Gamma, \Delta \vdash e : \tau}{\Gamma; \Delta \Vdash e : \tau}$$

The first one enables us to make a pure expression from a read-write expression if the read-write expression does not mutate any global variable. For example, though $\texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{end}$ is primarily considered as a read-write variable, if c_1 and c_2 do not assign to global variables, and it can be judged to be a pure expression. The second rule is obvious: it says we can regard a pure expression as a state-changing expression hence that the latter is a broader class of expressions. In other words, any expression is a state-changing expression.

The rules make it possible to judge the following expression, for example, be judged well-typed pure expression under $\Gamma = x : \mathsf{R}$:

$$42 + \left(\texttt{if}\ x \mathbin{\hat{>}} 0\ \texttt{then var}\ y := 42\ \texttt{in}\ y := 12; y\ \texttt{else}\ 42\right).$$

$$\frac{\Gamma; \cdot \Vdash e : \tau}{\Gamma \vdash e : \tau} \qquad \frac{\Gamma, \Delta \vdash e : \tau}{\Gamma; \Delta \Vdash e : \tau} \qquad \frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \qquad \frac{}{\Gamma \vdash \texttt{false} : \mathsf{B}} \qquad \frac{}{\Gamma \vdash \texttt{true} : \mathsf{B}} \qquad \frac{}{\Gamma \vdash k : \mathsf{Z}}$$

$$\frac{}{\Gamma \vdash \texttt{skip} : \mathsf{U}} \qquad \frac{\Gamma \vdash e : \mathsf{Z}}{\Gamma \vdash \iota(e) : \mathsf{R}} \qquad \frac{\Gamma \vdash e_1 : \mathsf{Z} \qquad \Gamma \vdash e_2 : \mathsf{Z}}{\Gamma \vdash e_1 \odot e_2 : \mathsf{Z}} \qquad \frac{\Gamma \vdash e_1 : \mathsf{R} \qquad \Gamma \vdash e_2 : \mathsf{R}}{\Gamma \vdash e_1 \boxdot e_2 : \mathsf{R}}$$

$$\frac{\Gamma \vdash e : \mathsf{R}}{\Gamma \vdash e^{-1} : \mathsf{R}} \qquad \frac{\Gamma \vdash e_1 : \mathsf{Z} \qquad \Gamma \vdash e_2 : \mathsf{Z}}{\Gamma \vdash e_1 < e_2 : \mathsf{B}} \qquad \frac{\Gamma \vdash e_1 : \mathsf{Z} \qquad \Gamma \vdash e_2 : \mathsf{Z}}{\Gamma \vdash e_1 = e_2 : \mathsf{B}} \qquad \frac{\Gamma \vdash e_1 : \mathsf{R} \qquad \Gamma \vdash e_2 : \mathsf{R}}{\Gamma \vdash e_1 \mathbin{\hat{<}} e_2 : \mathsf{B}}$$

$$\frac{\Gamma, x{:}\mathsf{Z} \vdash e : \mathsf{R}}{\Gamma \vdash (\texttt{lim}\ x.\ e) : \mathsf{R}} \qquad \frac{\Gamma; \Delta \Vdash c_1 : \mathsf{U} \qquad \Gamma; \Delta \Vdash c_2 : \tau}{\Gamma; \Delta \Vdash (c_1; c_2) : \tau} \qquad \frac{\Gamma, \Delta \vdash e : \sigma \qquad \Gamma; \Delta, x{:}\sigma \Vdash c : \tau}{\Gamma; \Delta \Vdash (\texttt{var}\ x := e\ \texttt{in}\ c) : \tau}$$

$$\frac{\Delta \vdash x : \tau \qquad \Gamma, \Delta \vdash e : \tau}{\Gamma; \Delta \Vdash (x := e) : \mathsf{U}} \qquad \frac{\Gamma, \Delta \vdash e : \mathsf{B} \qquad \Gamma; \Delta \Vdash c_1 : \tau \qquad \Gamma; \Delta \Vdash c_2 : \tau}{\Gamma; \Delta \Vdash (\texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{end}) : \tau}$$

$$\frac{\Gamma, \Delta \vdash e_i : \mathsf{B} \qquad \Gamma; \Delta \Vdash c_i : \tau \qquad (i = 1, \ldots, n)}{\Gamma; \Delta \Vdash (\texttt{case}\ e_1 \Rightarrow c_1 \mid \cdots \mid e_n \Rightarrow c_n\ \texttt{end}) : \tau} \qquad \frac{\Gamma, \Delta \vdash e : \mathsf{B} \qquad \Gamma; \Delta \Vdash c : \mathsf{U}}{\Gamma; \Delta \Vdash (\texttt{while}\ e\ \texttt{do}\ c\ \texttt{end}) : \mathsf{U}}$$

Figure 5.2: The typing rules of Clerical.

5.3 Denotational Semantics

5.3.1 Denotations of Data Types and Contexts

Denotational semantics provides a way of interpreting each part of a programming language by mapping it to a familiar mathematical object which is what we want to see it as ideally. For example,

we have R as a data type. However, what we really want to see from it is the set of real numbers $\mathbb{R}$. The *denotations of data types* in Clerical are defined as follows:

$$[\![\mathsf{U}]\!] \coloneqq \{*\} \qquad [\![\mathsf{B}]\!] \coloneqq \{tt, ff\} \qquad [\![\mathsf{Z}]\!] \coloneqq \mathbb{Z} \qquad [\![\mathsf{R}]\!] \coloneqq \mathbb{R}$$

The denotation of a context is the set of a mapping from each variable that is defined in the context to an actual mathematical value that the variable stores. Using the dependent product notation, the denotation of a typing context is defined by

$$[\![\Gamma]\!] \coloneqq \prod_{x \in \mathrm{dom}(\Gamma)} [\![\Gamma(x)]\!].$$

For *states* $\gamma \in [\![\Gamma]\!]$ and $\delta \in [\![\Delta]\!]$, we write $(\gamma, \delta) \in [\![\Gamma, \Delta]\!]$ for the join of the functions. And, we write $() \in [\![\cdot]\!]$ for the empty function. And, for a state $\gamma \in [\![\Gamma]\!]$, a variable $x \in \mathrm{dom}(\Gamma)$, and a value $v \in [\![\Gamma(x)]\!]$, we write $\gamma[x \mapsto v]$ to denote to the state whose function value at y is $\gamma(y)$ if $y \neq x$ and is v when $y = x$.

5.3.2 Semantic Construction

For a set S, define $S^{\mathsf{e}}_{\perp}$ be a poset on $S \cup \{\perp, \mathsf{e}\}$ with the ordering

$$x \leq y \quad :\Leftrightarrow \quad x = \perp \vee x = y \ .$$

We consider $\perp$ as nontermination and e as invalid computation[2]. We define a slightly modified Plotkin's powerdomain

$$\mathbb{P}_\star(S) \coloneqq \{X \subseteq S^{\mathsf{e}}_{\perp} \mid X \neq \emptyset \wedge (X \text{ infinite} \Rightarrow \perp \in X) \wedge (\mathsf{e} \in X \Rightarrow X = S^{\mathsf{e}}_{\perp})\}$$

which is ordered by Elgi-Milner ordering

$$X \sqsubseteq Y \quad :\Leftrightarrow \quad (\forall x \in X.\, \exists y \in Y.\, x \leq y) \wedge (\forall x \in Y.\, \exists x \in X.\, x \leq y) \ .$$

See that the order can be characterized by

$$X \sqsubseteq Y \quad \Longleftrightarrow \quad (\perp \in X \wedge X \subseteq Y \cup \{\perp\}) \vee X = Y \ .$$

Let us write $\mathfrak{b} = \{\perp\}$ and $\mathfrak{e} = S^{\mathsf{e}}_{\perp} \in \mathbb{P}_\star(S)$. We use the powerdomain to interpret our denotational semantics based on Kleene's fixed-point theorem. Hence, it is crucial to note the following.

Lemma 5.1. For any set S, the powerdomain $(\mathbb{P}_\star(S), \sqsubseteq)$ is a ω-CPO with a least element; any increasing chain in $\mathbb{P}_\star(S)$ has its limit in $\mathbb{P}_\star(S)$, and there is the least element in $\mathbb{P}_\star(S)$ which is $\mathfrak{b}$.

Proof. For any set S, $\mathbb{P}_\star(S)$ is exactly the Plotkin powerdomain except for $\mathfrak{e}$, which is above any element containing $\perp$. Suppose a chain in $\mathbb{P}_\star(S)$ does not encounter $\mathfrak{e}$. Then, the limit is the limit of the chain in the ordinary powerdomain. Suppose a chain contains $\mathfrak{e}$. Then, the limit of the chain is $\mathfrak{e}$. Since $\mathfrak{e}$ is above $\mathfrak{b}$, $\mathfrak{b}$ is the bottom element of $\mathbb{P}_\star(S)$. □

There is a rectifying operation ${}_\star : \{X \subseteq S^{\mathsf{e}}_{\perp}\} \to \mathbb{P}_\star(S)$ defined by

$$X_\star = \begin{cases} \mathfrak{e} & \text{if } e \in X, \\ X \cup \{\perp\} & \text{if } X \text{ infinite}, \\ \mathfrak{b} & \text{if } X = \emptyset, \\ X & \text{otherwise.} \end{cases}$$

[2] See that e and $\perp$ were dealt as $\perp$ in Chapter 3.

For an indexed set $f : I \to \mathbb{P}_\star(S)$, define the *join* operation

$$\biguplus_{x\in I} f(x) \coloneqq \left(\bigcup_{x\in I} f(x)\right)_\star$$

where $\biguplus_{x\in I} f(x) \in \mathbb{P}_\star(S)$.

To use this domain to interpret our denotational semantics, we need to define various lifting operations. Of course, instead of defining that ad hoc, we see that our powerdomain construction $S \mapsto \mathbb{P}_\star(S)$ is a moand in Set by taking a small detour:

Lemma 5.2. Consider an endofunctor $\mathsf{P} : \mathsf{Set} \to \mathsf{Set}$ defined on sets and functions as follows.

$$\begin{aligned} \mathsf{P}(A) &\coloneqq \{S \subseteq A \cup \{\bot\} \mid S \text{ infinite } \Rightarrow \bot \in S\} \\ \mathsf{P}(f : A \to B) &: S \mapsto \bigcup_{x\in S} \begin{cases} \{f(x)\} & \text{if } x \neq \bot \\ \{\bot\} & \text{otherwise.} \end{cases} \end{aligned}$$

(Note that the difference from the Plotkin powerdomain construction from Section 3.3 is that $\mathcal{P}$ allows the empty set.) The endofunctor with the collection of functions η_A, μ_A for each set A is a monad where they are defined by

$$\begin{aligned} \eta_A &: x \mapsto \{x\} \\ \mu_A &: S \mapsto \bigcup\!\!\cdot_{T\in S} \begin{cases} T & \text{if } T \neq \bot\,, \\ \{\bot\} & \text{otherwise}\,. \end{cases} \end{aligned}$$

Here, $\bigcup\!\!\cdot_{x\in I} f(x) \coloneqq \begin{cases} \emptyset & \text{if } \exists i \in I.\ f(i) = \emptyset\,, \\ \bigcup_{i\in I} f(i) & \text{otherwise}\,, \end{cases}$ for an indexed set $f : I \to \mathcal{P}(B)$.

Proof. We first need to show that $\eta : I \to \mathsf{P}$ and $\mu : \mathsf{P}^2 \to \mathsf{P}$ are indeed natural transformations. That is, the following diagrams commute:

$$\begin{array}{ccccc} A & \xrightarrow{\eta_A} & \mathsf{P}(A) & \xleftarrow{\mu_A} & \mathsf{P}^2(A) \\ \downarrow{\scriptstyle f} & & \downarrow{\scriptstyle \mathsf{P}(f)} & & \downarrow{\scriptstyle \mathsf{P}^2(f)} \\ B & \xrightarrow{\eta_B} & \mathsf{P}(B) & \xleftarrow{\mu_B} & \mathsf{P}^2(B) \end{array}$$

The left diagram is easy to verify that for any $x \in A$, $(\eta_B \circ f)\ (x) = \{f(x)\}$ and $(\mathsf{P}(f) \circ \eta_A)\ (x) = \mathsf{P}(f)(\{x\}) = \{f(x)\}$.

To verify the right diagram, for any $S \in \mathsf{P}^2(A)$, see that

$$\mathsf{P}^2(f)(S) = \bigcup_{T\in S} \begin{cases} \left\{ \bigcup_{x\in T} \begin{cases} \{f(x)\} & \text{if } x \neq \bot\,, \\ \{\bot\} & \text{otherwise}\,, \end{cases} \right\} & \text{if } T \neq \bot\,, \\ \{\bot\} & \text{otherwise}\,. \end{cases}$$

We can characterize the set as follows:

$$\begin{aligned} \emptyset = \mathsf{P}^2(f)(S) &\iff \emptyset = S\,, \\ \bot \in \mathsf{P}^2(f)(S) &\iff \bot \in S\,, \\ X \neq \bot \wedge X \in \mathsf{P}^2(f)(S) &\iff \exists T \in S.\ X = \bigcup_{x\in T} \begin{cases} \{f(x)\} & \text{if } x \neq \bot\,, \\ \{\bot\} & \text{otherwise}\,. \end{cases} \end{aligned}$$

Using the characterization, we can characterize $\mu_B(\mathsf{P}^2(f)(S))$ as follows:

$$\begin{aligned}
\mu_B(\mathsf{P}^2(f)(S)) = \emptyset &\iff S = \emptyset \vee \exists T \in S.\, \emptyset = \bigcup_{x \in T} \begin{cases} \{f(x)\} & \text{if } x \neq \bot \\ \{\bot\} & \text{otherwise} \end{cases} \\
&\iff S = \emptyset \vee \emptyset \in S\ , \\
y \in \mu_B(\mathsf{P}^2(f)(S)) &\iff (\neg(S = \emptyset \vee \emptyset \in S)) \wedge \big((y = \bot \wedge \bot \in S) \vee \exists X \in \mathsf{P}^2(f)(S).\, y \in X\big) \\
&\iff \emptyset \notin S \wedge \left((y = \bot \wedge \bot \in S) \vee \exists T \in S.\, y \in \bigcup_{x \in T} \begin{cases} \{f(x)\} & \text{if } x \neq \bot\ , \\ \{\bot\} & \text{otherwise}\ , \end{cases} \right) \\
&\iff \emptyset \notin S \wedge ((y = \bot \wedge (\bot \in S \vee \exists T \in S.\, \bot \in T)) \vee \exists T \in S.\, \exists x \in T.\, y = f(x))\ .
\end{aligned}$$

Now, see that $(\mathsf{P}(f) \circ \mu_A)(S)$ is

$$\bigcup_{x \in \left(\bigcup_{T \in S} \begin{cases} T & \text{if } T \neq \bot\ , \\ \{\bot\} & \text{otherwise}\ , \end{cases} \right)} \begin{cases} \{f(x)\} & \text{if } x \neq \bot\ , \\ \{\bot\} & \text{otherwise}\ . \end{cases}$$

It can be the emptyset if and only if $\bigcup\!\!\!\cdot_{T \in S} \begin{cases} T & \text{if } T \neq \bot\ , \\ \{\bot\} & \text{otherwise}\ , \end{cases}$ is the emptyset, and this happens exactly when $S = \emptyset$ or $\emptyset \in S$.

In the other case, assuming $\emptyset \notin S$, the set $\mathsf{P}(f)(\mu_A(S))$ contains $\bot$ if and only if $\bigcup\!\!\!\cdot_{T \in S} \begin{cases} T & \text{if } T \neq \bot\ , \\ \{\bot\} & \text{otherwise}\ , \end{cases}$ contains $\bot$. That is, when $\bot \in S$ or there is $T \in S$ such that $\bot \in T$.

Again assuming $\emptyset \notin S$, see that y which is not $\bot$ is in $(\mathsf{P}(f) \circ \mu_A)(S)$ if and only if there is $x \in \bigcup\!\!\!\cdot_{T \in S} \begin{cases} T & \text{if } T \neq \bot\ , \\ \{\bot\} & \text{otherwise}\ , \end{cases}$ such that $y = f(x)$. That is, if and only if there is $T \in S$ such that they x is in T.

We just observed that the two characterizations coincide, and that the collections of functions are indeed natural transformations.

Coherence Conditions

In order to confirm that the endofunctor P with two natural transformations $\eta : I \to \mathsf{P}$ and $\mu : \mathsf{P}^2 \to \mathsf{P}$ is a monad, we need to verify the monad laws which are the coherence conditions: (i) $\mu_A \circ \eta_{\mathsf{P}(A)} = \mathrm{id}_{\mathsf{P}(A)} = \mu_A \circ \mathsf{P}(\eta_A)$ and (ii) $\mu_A \circ \mu_{\mathsf{P}(A)} = \mu_A \circ \mathsf{P}(\mu_A)$ illustrated in the following diagrams.

$$\begin{array}{ccc}
\mathsf{P}(A) & \xrightarrow{\eta_{\mathsf{P}(A)}} & \mathsf{P}^2(A) \\
{\scriptstyle \mathsf{P}(\eta_A)}\downarrow & \searrow & \downarrow{\scriptstyle \mu_A} \\
\mathsf{P}^2(A) & \xrightarrow{\mu_A} & \mathsf{P}(A)
\end{array}
\qquad
\begin{array}{ccc}
\mathsf{P}^3(A) & \xrightarrow{\mu_{\mathsf{P}(A)}} & \mathsf{P}^2(A) \\
{\scriptstyle \mathsf{P}(\mu_A)}\downarrow & & \downarrow{\scriptstyle \mu_A} \\
\mathsf{P}^2(A) & \xrightarrow{\mu_A} & \mathsf{P}(A)
\end{array}$$

For the left diagram, see that for any $S \in \mathsf{P}(A)$,

$$(\mu_A \circ \eta_{\mathsf{P}(A)})(S) = \bigcup\!\!\!\cdot_{T \in \{S\}} \begin{cases} T & \text{if } T \neq \bot\ , \\ \{\bot\} & \text{otherwise}\ , \end{cases}$$

is simply S. And, for any $S \in \mathsf{P}(A)$, the set $(\mu_A \circ \mathsf{P}(\eta_A))(S)$ is defined as follows.

$$(\mu_A \circ \mathsf{P}(\eta_A))(S) = \mu_A \left(\bigcup_{x \in S} \begin{cases} \{\{x\}\} & \text{if } x \neq \bot \,, \\ \{\bot\} & \text{otherwise} \,, \end{cases} \right)$$

$$= \bigcup_{T \in \left(\bigcup_{x \in S} \begin{cases} \{\{x\}\} & \text{if } x \neq \bot \,, \\ \{\bot\} & \text{otherwise} \,, \end{cases} \right)} \begin{cases} T & \text{if } T \neq \bot \,, \\ \{\bot\} & \text{otherwise} \,. \end{cases}$$

It can be the emptyset if and only if $S = \emptyset$. In the other case, when $S \neq \emptyset$, see that $T \in \bigcup_{x \in S} \begin{cases} \{\{x\}\} & \text{if } x \neq \bot \\ \{\bot\} & \text{otherwise} \end{cases}$ if and only if $T = \bot \wedge \bot \in S$ or $T = \{x\} \wedge x \in S$. Hence, again, the above is precisely S.

For the second coherence condition, the diagram on the right, consider any $S \in \mathsf{P}^3(A)$. See that

$$\mu_{\mathsf{P}(A)}(S) = \bigcup_{T \in S} \begin{cases} T & \text{if } T \neq \bot \,, \\ \{\bot\} & \text{otherwise} \,, \end{cases}$$

$$\mu_A(\mu_{\mathsf{P}(A)}(S)) = \bigcup_{R \in \left(\bigcup_{T \in S} \begin{cases} T & \text{if } T \neq \bot \,, \\ \{\bot\} & \text{otherwise} \,, \end{cases} \right)} \begin{cases} R & \text{if } R \neq \bot \,, \\ \{\bot\} & \text{otherwise} \,. \end{cases}$$

and

$$\mathsf{P}(\mu_A)(S) = \bigcup_{T \in S} \begin{cases} \left\{ \bigcup_{R \in T} \begin{cases} R & \text{if } R \neq \bot \,, \\ \{\bot\} & \text{otherwise} \,, \end{cases} \right\} & \text{if } T \neq \bot \,, \\ \{\bot\} & \text{otherwise} \,, \end{cases}$$

$$\mu_A(\mathsf{P}(\mu_A)(S)) = \bigcup_{P \in \left(\bigcup_{T \in S} \begin{cases} \left\{ \bigcup_{R \in T} \begin{cases} R & \text{if } R \neq \bot \,, \\ \{\bot\} & \text{otherwise} \,, \end{cases} \right\} & \text{if } T \neq \bot \,, \\ \{\bot\} & \text{otherwise} \,, \end{cases} \right)} \begin{cases} P & \text{if } P \neq \bot \,, \\ \{\bot\} & \text{otherwise} \,. \end{cases}$$

To ease the presentation, let us write $X \coloneqq \mu_A(\mu_{\mathsf{P}(A)}(S))$ and $Y \coloneqq \mu_A(\mathsf{P}(\mu_A)(S))$.

See that $X = \emptyset$ if and only if (i) $S = \emptyset$, (ii) $\emptyset \in S$, or (iii) $\emptyset \in \bigcup_{T \in S} \begin{cases} T & \text{if } T \neq \bot \,, \\ \{\bot\} & \text{otherwise} \,. \end{cases}$ The third condition holds if and only if $\exists Q \in S$ s.t. $\emptyset \in Q$ and $\emptyset \notin S$.

On the other hand, $Y = \emptyset$ if and only if (i) $S = \emptyset$, or (ii) $\emptyset \in \bigcup_{T \in S} \begin{cases} \left\{ \bigcup_{R \in T} \begin{cases} R & \text{if } R \neq \bot \,, \\ \{\bot\} & \text{otherwise} \,, \end{cases} \right\} & \text{if } T \neq \bot \,, \\ \{\bot\} & \text{otherwise} \,. \end{cases}$ The second condition holds if and only if there is $T \in S$ such that $\bigcup_{R \in T} \begin{cases} R & \text{if } R \neq \bot \,, \\ \{\bot\} & \text{otherwise} \,. \end{cases}$ is the emptyset. And, this condition holds if and only if $T = \emptyset$ or $\emptyset \in T$. Therefore, $X = \emptyset$ if and only if $Y = \emptyset$.

From this point, assume $X, Y \neq \emptyset$.

See that $\bot \in X$ if and only if $\bot \in \biguplus_{T \in S} \begin{cases} T & \text{if } T \neq \bot\,, \\ \{\bot\}\,, & \text{otherwise} \end{cases}$ or there is R such that $\bot \in R$ and $R \in \biguplus_{T \in S} \begin{cases} T & \text{if } T \neq \bot\,, \\ \{\bot\} & \text{otherwise}\,. \end{cases}$ This holds if and only if $\bot \in S$, there is $T \in S$ such that $\bot \in T$, or there is R, T such that $\bot \in R$, $R \in T$, and $T \in S$.

And, $\bot \in Y$ holds if and only if $\bot \in \bigcup_{T \in S} \begin{cases} \left\{ \biguplus_{R \in T} \begin{cases} R & \text{if } R \neq \bot\,, \\ \{\bot\} & \text{otherwise}\,, \end{cases} \right\} & \text{if } T \neq \bot\,, \\ \{\bot\} & \text{otherwise}\,, \end{cases}$ or there is P such that $\bot \in P$ and $P \in \bigcup_{T \in S} \begin{cases} \left\{ \biguplus_{R \in T} \begin{cases} R & \text{if } R \neq \bot\,, \\ \{\bot\} & \text{otherwise}\,, \end{cases} \right\} & \text{if } T \neq \bot\,, \\ \{\bot\} & \text{otherwise}\,. \end{cases}$ The first holds if and only if $\bot \in S$. And, the second holds if and only if there is $T \in S$ such that there is $\bot \in T$ or there is R such that $R \in T$ and $\bot \in R$. Therefore, $\bot \in X$ if and only if $\bot \in Y$.

Now, suppose any x that is not $\bot$. See that $x \in X$ if and only if there is R such that $x \in R$ and $R \in \biguplus_{T \in S} \begin{cases} T & \text{if } T \neq \bot\,, \\ \{\bot\} & \text{otherwise}\,. \end{cases}$ This holds if and only if there is R, T such that $x \in R$, $R \in T$, and $T \in S$.

And, $x \in Y$ holds if and only if there is P such that $x \in P$ and

$$P \in \bigcup_{T \in S} \begin{cases} \left\{ \biguplus_{R \in T} \begin{cases} R & \text{if } R \neq \bot\,, \\ \{\bot\} & \text{otherwise}\,, \end{cases} \right\} & \text{if } T \neq \bot\,, \\ \{\bot\} & \text{otherwise}\,. \end{cases}$$

This holds if and only if there is $T \in S$ such that there is R such that $R \in T$ and $x \in R$. Therefore, $x \in X$ if and only if $x \in Y$.

Now we conclude $\mu_A \circ \mu_{\mathsf{P}(A)} = \mu_A \circ \mathsf{P}(\mu_A)$ holds. Satisfying the coherence conditions, (P, η, μ) is a monad. □

Corollary 5.1. The powerdomain construction $S \mapsto \mathbb{P}_\star(S)$ as an endofunctor in Set whose action on functions is defined by

$$\mathbb{P}_\star(f : A \to B) \coloneqq S \mapsto \bigcup_{x \in S} \begin{cases} \mathfrak{e} & \text{if } x = \mathsf{e}, \\ \{\bot\} & \text{if } x = \bot, \\ \{f(x)\} & \text{otherwise}, \end{cases}$$

is a monad with the unit $\eta_A : x \mapsto \{x\}$ and the multiplication

$$\mu_A : S \mapsto \biguplus_{T \in S} \begin{cases} \{\mathsf{e}\} & \text{if } T = \mathsf{e}, \\ \{\bot\} & \text{if } T = \bot, \\ T & \text{otherwise}. \end{cases}$$

Proof. For any S, see that $\mathsf{P}(S) \to \mathbb{P}_\star(S)$ by

$$(f_S : \mathsf{P}(S) \to \mathbb{P}_\star(S)) : X \mapsto \begin{cases} X & \text{if } X \neq \emptyset \\ \mathfrak{e} & \text{otherwise}. \end{cases}$$

is an isomorphism where

$$(g_S : \mathbb{P}_\star(S) \to \mathsf{P}(S)) : Y \mapsto \begin{cases} \emptyset & \text{if } Y \neq \mathfrak{e} \\ Y & \text{otherwise} \end{cases}$$

is its inverse. Hence, $\mathbb{P}_\star$ also is a moand with the unit

$$f_S \circ \eta^{\mathsf{P}}_S : S \to \mathbb{P}_\star(S),$$

and the multiplication

$$f_S \circ \mu^{\mathsf{P}}_S \circ \mathsf{P}(g_S) \circ g_{\mathbb{P}_\star(S)} : \mathbb{P}_\star(\mathbb{P}_\star(S)) \to \mathbb{P}_\star(S)$$

□

The construction $\mathbb{P}_\star$ being a monad on Set, it is a strong monad with the tensorial strength:

$$\begin{array}{rcccl} \beta_{S,T} & : & S \times \mathbb{P}_\star(T) & \to & \mathbb{P}_\star(S \times T) \\ & : & (x, Y) & \mapsto & \biguplus_{y \in Y} \begin{cases} \{\mathsf{e}\} & \text{if } y = \mathsf{e}, \\ \{\bot\} & \text{if } y = \bot, \\ (x, y) & \text{otherwise.} \end{cases} \end{array}$$

And, the natural transformation is obtained from the strength:

$$\begin{array}{rcccl} \alpha_{S,T} & : & \mathbb{P}_\star(S) \times \mathbb{P}_\star(T) & \to & \mathbb{P}_\star(S \times T) \\ & : & (X, Y) & \mapsto & \biguplus_{x \in X \wedge y \in Y} \begin{cases} \mathfrak{e} & \text{if } x = \mathsf{e} \vee y = \mathsf{e}, \\ \{\bot\} & \text{if } x = \bot \vee y = \bot, \\ (x, y) & \text{otherwise.} \end{cases} \end{array}$$

Also, it is countably applicative with θ defined by

$$\begin{array}{rcccl} \theta_S & : & (\mathbb{N} \to \mathbb{P}_\star(S)) & \to & \mathbb{P}_\star(\mathbb{N} \to S) \\ & : & (n \mapsto X_n) & \mapsto & \biguplus_{x_i \in X_i} \begin{cases} \{\mathsf{e}\} & \text{if } \exists i.\ \mathsf{e} = x_i, \\ \{\bot\} & \text{if } \exists i.\ \bot = x_i, \\ \{n \mapsto x_n\} & \text{otherwise.} \end{cases} \end{array}$$

Hence, we can define various types of lifting.

- When $f : A_1 \times \cdots \times A_d \to B$, we can lift it to $f^\dagger : \mathbb{P}_\star(A_1) \times \cdots \times \mathbb{P}_\star(A_d) \to \mathbb{P}_\star(B)$ by consecutively precomposing appropriate α on $\mathbb{P}_\star(f)$. It happens to be

$$f^\dagger(S_1, \cdots, S_d) = \biguplus_{(x_1, \cdots, x_d) \in S_1 \times \cdots \times S_d} \begin{cases} \{\mathsf{e}\} & \text{if } \exists i.\ x_i = \mathsf{e}, \\ \{\bot\} & \text{if } \exists i.\ x_i = \bot, \\ \{f(x_1, \cdots, x_d)\} & \text{otherwise.} \end{cases}$$

- When $f : A_1 \times \cdots \times A_d \to \mathbb{P}_\star(B)$, we can lift it to $f^\dagger : \mathbb{P}_\star(A_1) \times \cdots \times \mathbb{P}_\star(A_d) \to \mathbb{P}_\star(B)$ by consecutively precomposing appropriate α on $\mu_B \circ \mathbb{P}_\star(f)$. It happens to be

$$f^\dagger(S_1, \cdots, S_d) = \biguplus_{(x_1, \cdots, x_d) \in S_1 \times \cdots \times S_d} \begin{cases} \{\mathsf{e}\} & \text{if } \exists i.\ x_i = \mathsf{e}, \\ \{\bot\} & \text{if } \exists i.\ x_i = \bot, \\ f(x_1, \cdots, x_d) & \text{otherwise.} \end{cases}$$

- When $f : A_1 \times \cdots \times A_d \to B$, we can lift it to $f^{\dagger_i} : A_1 \times \mathbb{P}_\star(A_i) \cdots \times A_d \to \mathbb{P}_\star(B)$ by precomposing appropriate β on $\mathbb{P}_\star(f)$. It happens to be

$$f^\dagger(x_1, \cdots, S_i, \cdots x_d) = \biguplus_{x_i \in S_i} \begin{cases} \{\mathsf{e}\} & \text{if } x_i = \mathsf{e}, \\ \{\bot\} & \text{if } x_i = \bot, \\ \{f(x_1, \cdots, x_i, \cdots x_d)\} & \text{otherwise.} \end{cases}$$

- When $f : A_1 \times \cdots \times A_d \to \mathbb{P}_\star(B)$, we can lift it to $f^{\dagger_i} : A_1 \times \mathbb{P}_\star(A_i) \cdots \times A_d \to \mathbb{P}_\star(B)$ by precomposing appropriate β on $\mu_B \circ \mathbb{P}_\star(f)$. It happens to be

$$f^\dagger(x_1, \cdots, S_i, \cdots x_d) = \biguplus_{x_i \in S_i} \begin{cases} \{\mathsf{e}\} & \text{if } x_i = \mathsf{e}, \\ \{\bot\} & \text{if } x_i = \bot, \\ f(x_1, \cdots, x_i, \cdots x_d) & \text{otherwise.} \end{cases}$$

- When $f : (\mathbb{N} \to A) \to B$, we can lift it to $f^\dagger : (\mathbb{N} \to \mathbb{P}_\star(A)) \to \mathbb{P}_\star(B)$ by precomposing θ_A on $\mathbb{P}_\star(f)$. It happens to be

$$f^\dagger((S_i)_{i \in \mathbb{N}}) = \biguplus_{x_i \in S_i} \begin{cases} \{\mathsf{e}\} & \text{if } \exists i.\ x_i = \mathsf{e}, \\ \{\bot\} & \text{if } \exists i.\ x_i = \bot, \\ \{f((x_i)_{i \in \mathbb{N}})\} & \text{otherwise.} \end{cases}$$

- When $f : (\mathbb{N} \to A) \to \mathbb{P}_\star(B)$, we can lift it to $f^\dagger : (\mathbb{N} \to \mathbb{P}_\star(A)) \to \mathbb{P}_\star(B)$ by precomposing θ_A on $\mu_B \circ \mathbb{P}_\star(f)$. It happens to be

$$f^\dagger((S_i)_{i \in \mathbb{N}}) = \biguplus_{x_i \in S_i} \begin{cases} \{\mathsf{e}\} & \text{if } \exists i.\ x_i = \mathsf{e}, \\ \{\bot\} & \text{if } \exists i.\ x_i = \bot, \\ f((x_i)_{i \in \mathbb{N}}) & \text{otherwise.} \end{cases}$$

For a mapping $f : A_1 \times \cdots \times A_d \to B$ and a set $X \in \mathbb{P}_\star(S_i)$, define

$$(\mathsf{let}\ x_i \leftarrow X\ \mathsf{in}\ f) \coloneqq f^{\dagger_i}|_{S_i := X} : A_1 \times \cdots A_{i-1} \times A_{i+1} \times \cdots A_d \to \mathbb{P}_\star(B).$$

Similarly, for a mapping $f : A_1 \times \cdots \times A_d \to \mathbb{P}_\star(B)$ and a set $X \in \mathbb{P}_\star(S_i)$, define

$$(\mathsf{let}\ x_i \leftarrow X\ \mathsf{in}\ f) \coloneqq f^{\dagger_i}|_{S_i := X} : A_1 \times \cdots A_{i-1} \times A_{i+1} \times \cdots A_d \to \mathbb{P}_\star(B).$$

For a function valued function $f : A_1 \to \cdots \to A_d \to B$, let us define $f^\dagger$ and $f^{\dagger_i}$ by $\mathsf{curry}((\mathsf{uncurry}(f))^\dagger)$ and $\mathsf{curry}((\mathsf{uncurry}(f))^{\dagger_i})$.

Let us define an *auxiliary* lifting. For a mapping $f : S \to T_\bot^{\mathsf{e}}$, define its codomain lifting $f^\ddagger : S \to \mathbb{P}_\star(T)$ by

$$f^\ddagger(x) = \{f(x)\}_\star.$$

Now, we see their domain-theoretic properties.

Lemma 5.3. Lifting is monotone in both arguments that

1. for any $f : S \to \mathbb{P}_\star(T)$ and $X, Y \in \mathbb{P}_\star(S)$, if $X \sqsubseteq Y$, then $f^\dagger(X) \sqsubseteq f^\dagger(Y)$, and
2. for any $f, g : S \to \mathbb{P}_\star(T)$ and $X \in \mathbb{P}_\star(S)$, if $\forall x \in X.\ f(x) \sqsubseteq g(x)$ then $f^\dagger(X) \sqsubseteq g^\dagger(X)$.

Proof. (Proof of 1). Suppose $\mathsf{e} \in X$. Then, $f^\dagger(X) = f^\dagger(Y) = \mathfrak{e}$. Hence, assume $\mathsf{e} \notin X$. If $\mathsf{e} \in Y$, in order to make $X \sqsubseteq Y$ hold, $\bot \in X$ holds. Hence, $\bot \in f^\dagger(X) \sqsubseteq \mathfrak{e} = f^\dagger(Y) = \mathfrak{e}$. Further suppose $\mathsf{e} \notin Y$ too. If $\bot \in X$ and $\bot \in Y$, $X' \subseteq Y'$ where $X = X' \cup \{\bot\}$, $Y = Y' \cup \{\bot\}$, and $X, Y \subseteq S$. In the case, $f^\dagger(X') \subseteq f^\dagger(Y')$ and $f^\dagger(X) = f^\dagger(X') \cup \{\bot\}$ and $f^\dagger(Y) = f^\dagger(Y') \cup \{\bot\}$. Therefore, $f^\dagger(X) \sqsubseteq f^\dagger(Y)$. Now, suppose $\bot \notin Y$. In the case, $f^\dagger(X) = f^\dagger(X') \cup \{\bot\} \subseteq f^\dagger(Y) \cup \{\bot\}$. Hence, $f^\dagger(X) \sqsubseteq f^\dagger(Y)$. If $\bot \notin X$, $X = Y$.
(Proof of 2). If $\mathsf{e} \in X$, $f^\dagger(X) = g^\dagger(X)$. Hence, suppose $\mathsf{e} \notin X$.

See that for each $x \in X$, $f(x) \setminus \{\bot\} \subseteq g(x) \setminus \{\bot\}$ holds. And, if $\bot \in f^\dagger(X)$, it holds that $f^\dagger(X) = \{\bot\} \cup \bigcup_{x \in X} f(x) \setminus \{\bot\}$. And, similarly. $g^\dagger(X) = \{\bot\} \cup \bigcup_{x \in X} g(x) \setminus \{\bot\}$ if $\bot \in g^\dagger(X)$. Hence, due to the monotonicity of the set union, we have $f^\dagger(X) \sqsubseteq g^\dagger(X)$.

Suppose $\bot \notin f^\dagger(X)$. It holds only if for each $x \in X$, it holds that $\bot \notin f(x)$. As $f(x) \sqsubseteq g(x)$, it holds that $f(x) = g(x)$ for all x.

Now suppose $\bot \in f^\dagger(X)$ and $\bot \notin g^\dagger(x)$. Then, $f^\dagger(X) = \{\bot\} \cup \bigcup_{x \in X} f(x) \setminus \{\bot\}$ and $g^\dagger(X) = \bigcup_{x \in X} g(x)$. And, for each $x \in X$, it holds that $f(x) \setminus \{\bot\} \subseteq g(x) \setminus \{\bot\} = g(x)$. Hence, we have $f^\dagger(X) \sqsubseteq g^\dagger(X)$.

□

Lemma 5.4. Lifting is continuous in both arguments that

1. for any $f : S \to \mathbb{P}_\star(T)$ and $X_i \in \mathbb{P}_\star(S)$ where $(X_i)_{i \in \mathbb{N}}$ is a chain with regards to the point-wise ordering,
$$\bigsqcup_{i \in \mathbb{N}} f^\dagger(X_i) = f^\dagger\Big(\bigsqcup_{i \in \mathbb{N}} X_i\Big).$$
holds. And,

2. for any $f_i : S \to \mathbb{P}_\star(T)$ and $X \in \mathbb{P}_\star(S)$ where $(f_i)_{i \in \mathbb{N}}$ is a chain with regards to the point-wise ordering,
$$\bigsqcup_{i \in \mathbb{N}} f_i^\dagger(X) = \Big(\bigsqcup_{i \in \mathbb{N}} f_i\Big)^\dagger(X).$$
holds.

Proof. (Proof of 1): The fact that $(f^\dagger(X_i))_{i \in \mathbb{N}}$ is a chain is direct from Lemma 5.3.1. Let us write X for the limit of $(X_i)_{i \in \mathbb{N}}$.

See that $\mathsf{e} \in \bigsqcup_{i \in \mathbb{N}} f^\dagger(X_i)$ if and only if $\exists i.\ \mathsf{e} \in f^\dagger(X_i)$ if and only if $\exists i.\ \mathsf{e} \in X_i \vee \exists x \in X_i.\ \mathsf{e} \in f(x)$. And, $\mathsf{e} \in f^\dagger(X)$ if and only if $\mathsf{e} \in X \vee \exists x \in X.\ \mathsf{e} \in f(x)$ if and only if $(\exists i.\ \mathsf{e} \in X_i) \vee (\exists i.\ x \in X_i.\ \mathsf{e} \in f(x))$. And, the two conditions are equivalent.

Similarly, assuming e is not in the both sides, $\bot \in \bigsqcup_{i \in \mathbb{N}} f^\dagger(X_i)$ if and only if $\forall i.\ \bot \in f^\dagger(X_i)$ if and only if $\forall i.\ \bot \in X_i \vee \exists x \in X_i.\ \bot \in f(x)$. And, $\bot \in f^\dagger(X)$ if and only if $\bot \in X \vee \exists x \in X.\ \bot \in f(x)$ if and only if $(\forall i.\ \bot \in X_i) \vee \exists i.\ \exists x \in X_i.\ \bot \in f(x)$.

If $(\forall i.\ \bot \in X_i)$, we immediately have $\forall i.\ \bot \in X_i \vee \exists x \in X_i.\ \bot \in f(x)$. Suppose $\exists i.\ \exists x \in X_i.\ \bot \in f(x)$. Then, since X_i is a chain all X_j with $j \geq i$ contains x. Therefore, $f^\dagger(X_j)$ for all $j \geq i$ contains $\bot$. Hence, $\bot \in \bigsqcup_{i \in \mathbb{N}} f^\dagger(X_i)$.

If $\forall i.\ \bot \in X_i \vee \exists x \in X_i.\ \bot \in f(x)$, if there is at least one i such the right term $\exists x \in X_i.\ \bot \in f(x)$ holds, then since the x such that $\bot \in f(x)$ is in X, $\bot \in f^\dagger(X)$. Hence, suppose there is no such i. That is, $\forall i.\ \bot \in X_i$ holds. Then, $\bot \in f^\dagger(X)$ as well.

Now, suppose $\bot$ is not in the both sides as well. Then, for any $y \in T$, see that $y \in \bigsqcup_{i\in\mathbb{N}} f^\dagger(X_i)$ if and only if there is $x \in T$ in X_i where $y \in f(x)$. And, $y \in f^\dagger(X)$ if and only if there is $x \in T$ in X where $y \in f(x)$. See that the two conditions are equivalent.

(Proof of 2): The fact that $(f_i^\dagger(X))_{i\in\mathbb{N}}$ is a chain is direct from Lemma 5.3.2. Let us write f for the limit of $(f_i)_{i\in\mathbb{N}}$ with regards to the point-wise ordering.

Observe that $\mathsf{e} \in \bigsqcup_{i\in\mathbb{N}} f_i^\dagger(X)$ if and only if $\exists i.\ \mathsf{e} \in f_i^\dagger(X)$ if and only if $\mathsf{e} \in X \vee \exists x \in X.\ \exists i.\ \mathsf{e} \in f_i(x)$. On the other hand, $\mathsf{e} \in f^\dagger(X)$ if and only if $\mathsf{e} \in X \vee \exists x \in X.\ \mathsf{e} \in f(x)$ if and only if $\mathsf{e} \in X \vee \exists x \in X.\ \exists i.\ \mathsf{e} \in f_i(x)$.

Now, suppose e is not contained in the both sides. Then, $\bot \in \bigsqcup_{i\in\mathbb{N}} f_i^\dagger(X)$ if and only if $\forall i.\ \bot \in f_i^\dagger(X)$ if and only if $\forall i.\ \bot \in X \vee \exists x \in X.\ \bot \in f_i(x)$ if and only if $\bot \in X \vee \forall i.\ \exists x \in X.\ \bot \in f_i(x)$. And, $\bot \in f^\dagger(X)$ if and only if $\bot \in X \vee \exists x \in X.\ \bot \in f(x)$ if and only if $\bot \in X \vee \exists x \in X.\ \forall i.\ \bot \in f_i(x)$.

Since the case $\bot \in X$ is obvious, suppose $\bot \notin X$. Then, $\bot \in \bigsqcup_{i\in\mathbb{N}} f_i^\dagger(X)$ if and only if $\forall i.\ \exists x \in X.\ \bot \in f_i(x)$. Let $x_i \in X$ satisfies $\bot \in f_i(x_i)$ for all i. Since X is finite, there is $x \in X$ that is x_i for infinitely many i. Since $\bot \in f_i(x)$ implies $\bot \in f_j(x)$ for all $j \leq i$, $\bot \in f_i(x)$ for all i. Therefore, $\bot \in \bigsqcup_{i\in\mathbb{N}} f_i^\dagger(X)$ if and only if $\exists x \in X.\ \forall i.\ \bot \in f_i(x)$ if and only if $\bot \in f^\dagger(X)$.

Now, suppose $\bot$ also is not contained in the both sides. For any $y \in T$, see that $y \in \bigsqcup_{i\in\mathbb{N}} f_i^\dagger(X)$ if and only if $\exists i.\ y \in f_i^\dagger(X)$ if and only if $\exists i.\ \exists x \in X.\ y \in f_i(x)$. And, $y \in f(X)$ if and only if $\exists x \in X.\ y \in f(x)$ if and only if $\exists x \in X.\ \exists i.\ y \in f_i(x)$. Hence, we conclude that the desired equation holds. □

Lemma 5.5. Set union is continuous in that for any $X_i, Y \in \mathbb{P}_\star(S)$ where $(X_i)_{i\in\mathbb{N}}$ is a chain, the following holds:

$$\bigsqcup_{i\in\mathbb{N}}(X_i \cup Y) = \left(\bigsqcup_{i\in\mathbb{N}} X_i\right) \cup Y.$$

Proof. The fact that $(X_i \cup Y)$ is also a chain is direct from the definition of the Egli-Milner ordering. Let us write X for the limit of $(X_i)_{i\in\mathbb{N}}$.

See that $\mathsf{e} \in X \cup Y$ if and only if $\mathsf{e} \in X \vee \mathsf{e} \in Y$ if and only if $(\exists i.\ \mathsf{e} \in X_i) \vee \mathsf{e} \in Y$ if and only if $\exists i.\ (\mathsf{e} \in X_i \vee \mathsf{e} \in Y)$ if and only if $\mathsf{e} \in \bigsqcup_{i\in\mathbb{N}}(X_i \cup Y)$.

Now, assume e is not in the both sides. Then, $\bot \in X \cup Y$ if and only if $\bot \in X \vee \bot \in Y$ if and only if $(\forall i.\ \bot \in X_i) \vee \bot \in Y$. And, $\bot \in \bigsqcup_{i\in\mathbb{N}}(X_i \cup Y)$ if and only if $\forall i.\ (\bot \in X_i \vee \bot \in Y)$ if and only if $(\forall i.\ \bot \in X_i) \vee \bot \in Y$.

For the last case, assuming $\bot$ is also not in the both sides, for any $x \in S$, $x \in X \cup Y$ if and only if $x \in X \vee x \in Y$ if and only if $(\exists i.\ x \in X_i) \vee x \in Y$. And, $x \in \bigsqcup_{i\in\mathbb{N}}(X_i \cup Y)$ if and only if $\exists i.\ (x \in X_i \vee x \in Y)$ if and only if $(\exists i.\ x \in X_i) \vee x \in Y$. □

Lemma 5.6. The Kleisli composition are continuous in both arguments that for any $f_i, g : S \to \mathbb{P}_\star(S)$ where $(f_i)_{i\in\mathbb{N}}$ is a chain with regards to the point-wise ordering, the following equations holds:

1. $\left(\bigsqcup_{i\in\mathbb{N}} f_i\right)^\dagger \circ g = \bigsqcup_{i\in\mathbb{N}} \left(f_i^\dagger \circ g\right)$,
2. $g^\dagger \circ \bigsqcup_{i\in\mathbb{N}} f_i = \bigsqcup_{i\in\mathbb{N}} g^\dagger \circ f_i$.

Proof. They are direct from Lemma 5.5. □

5.3.3 Denotations of Expressions

For a well-typed pure expression $\Gamma \vdash e : \tau$, its denotation on a state $\gamma \in [\![\Gamma]\!]$ is the set of values that e evaluates to regarding nondeterminism. The case $\bot \in [\![\Gamma \vdash e : \tau]\!]\gamma$ denotes that a nondeterministic branch in evaluating e under γ leads nontermination. And, $\mathtt{e} \in [\![\Gamma \vdash e : \tau]\!]\gamma$ denotes that there is a nondeterministic branch facing total failure.

Similarly, for a well-typed read-write expression $\Gamma; \Delta \Vdash c : \tau$, its denotation on γ, δ is a set of $(\delta', v) \in [\![\Delta]\!] \times [\![\tau]\!]$ representing that there is a nondeterministic branch in the execution that leads to new read-write state δ' with v being evaluated. There also are other cases that $\mathtt{e}$ and $\bot$ are in the denotation which denotes the same cases for pure expressions.

Hence, the meaning of a pure expression $\Gamma \vdash e : \tau$ is a map

$$[\![\Gamma \vdash e : \tau]\!] : [\![\Gamma]\!] \to \mathbb{P}_\star([\![\tau]\!])$$

and the meaning of a read-write expression $\Gamma; \Delta \Vdash c : \tau$ is a map

$$[\![\Gamma; \Delta \Vdash c : \tau]\!] : [\![\Gamma]\!] \to [\![\Delta]\!] \to \mathbb{P}_\star([\![\Delta]\!] \times [\![\tau]\!]) \,.$$

Let us recall and define some auxiliary functions. The first is conditional:

$$\begin{array}{rcccl} \mathsf{Cond}_S & : & 2 \times S \times S & \to & S \\ & := & (b, x, y) & \mapsto & \begin{cases} x & \text{if } b = tt \\ y & \text{otherwise.} \end{cases} \end{array}$$

The second is to interpret guarded nondeterminisms:

$$\begin{array}{rcccl} \mathsf{Guard}_S & : & 2 \times \mathbb{P}_\star(S) & \to & \mathcal{P}(S^{\mathsf{e}}_\bot) \\ & := & (b, X) & \mapsto & \begin{cases} X & \text{if } b = tt \\ \emptyset & \text{otherwise.} \end{cases} \end{array}$$

In order to interpret `lim`, we take $\mathtt{e}$ extension of the partial limit function lim. That is,

$$\mathsf{lim}((x_i)_{i\in\mathbb{N}}) \lfloor_{\mathsf{e}} = \begin{cases} \mathsf{lim}((x_i)_{i\in\mathbb{N}}) & \text{if } \exists x.\, \forall k \in \mathbb{Z}.\, |x - x_k| \leq 2^{-k} \\ \mathsf{e} & \text{otherwise.} \end{cases}$$

a function from $\mathbb{R}^\omega \to \mathbb{R}^{\mathsf{e}}_\bot$. And, define $\mathsf{E}_S : \mathbb{P}_\star(S) \to \mathbb{P}_\star(S)$ by

$$\mathsf{E}_S(X) := \begin{cases} \mathfrak{e} & \text{if } \bot \in X, \\ X & \text{otherwise.} \end{cases}$$

See that

$$\mathsf{E}_{\mathbb{R}} \circ (\mathsf{lim} \lfloor_{\mathsf{e}})^{\ddagger\dagger}(X_i)_{i\in\mathbb{N}} = \begin{cases} \{x\} & \text{if } \forall i.\, X_i \subseteq \mathbb{R} \wedge \forall y \in X_i \,.\, |x - y| \leq 2^{-i}, \\ \mathfrak{e} & \text{otherwise.} \end{cases}$$

For the real number comparison $\hat{<}$, we consider the $\bot$ extension of the partial comparison $<$:

$$\square_1 < \lfloor_\bot \square_2 : (x, y) \mapsto \begin{cases} tt & \text{if } x < y, \\ ff & \text{if } y < x, \\ \bot & \text{otherwise,} \end{cases}$$

Similarly for $^{-1}$, we take $\bot$ extension of the partial function $\square^{-1}$:

$$\square^{-1\downarrow_\bot} : \mathbb{R} \ni x \mapsto \begin{cases} x^{-1} & \text{if } x \neq 0, \\ \bot & \text{otherwise.} \end{cases}$$

For a set T, let us write $j : T \times \{*\} \to T$ for the projection map and $k : T \to T \times \{*\}$ for the map $x \mapsto (x, *)$. For any $b : T \to \mathbb{P}_\star(2)$ and $c : T \to \mathbb{P}_\star(T \times \{*\})$, let us define

$$\mathsf{W}(b, c) : (f : T \to \mathbb{P}_\star(T \times \{*\})) \mapsto \mathsf{Cond}^{\dagger_1}_{\mathbb{P}_\star(T\times\{*\})} \circ \big(b \times (f^\dagger \circ j^\dagger \circ c) \times k^\dagger\big).$$

Lemma 5.7. For any $b : T \to \mathbb{P}_\star(2)$ and $c : T \to \mathbb{P}_\star(T \times \{*\})$, the mapping $\mathsf{W}(b, c)$ is continuous; i.e., for any chain $(f_i)_{i\in\mathbb{N}}$ w.r.t. the point-wise ordering,

$$\bigsqcup_{i\in\mathbb{N}} \mathsf{W}(b, c)(f_i) = \mathsf{W}(b, c)\Big(\bigsqcup_{i\in\mathbb{N}} f_i\Big).$$

Proof. For any mapping $\bar{b} : T \to \mathbb{P}_\star(2)$ and $\bar{c} : T \to \mathbb{P}_\star(T)$, define the mapping

$$\bar{\mathsf{W}}_{\bar{b},\bar{c}} : (\bar{f} : T \to \mathbb{P}_\star(T)) \mapsto \mathsf{Cond}^{\dagger_1}_{\mathbb{P}_\star(T)} \circ (\bar{b} \times \bar{f}^\dagger \circ \bar{c} \times \eta).$$

and see that

$$\mathsf{W}_{b,c}(f) = k^\dagger \circ \bar{\mathsf{W}}_{j^\dagger\circ b, j^\dagger\circ c}(j^\dagger \circ f)$$

holds. The mapping $\bar{\mathsf{W}}$ is continuous by the definition of Cond, Lemma 5.4, and Lemma 5.5.

Hence, it holds that

$$\begin{aligned} \mathsf{W}_{b,c}\Big(\bigsqcup\nolimits_{i\in\mathbb{N}} f_i\Big) &= k^\dagger \circ \bar{\mathsf{W}}_{j^\dagger\circ b, j^\dagger\circ c}\Big(j^\dagger \circ \bigsqcup\nolimits_{i\in\mathbb{N}} f_i\Big) \\ &= k^\dagger \circ \bar{\mathsf{W}}_{j^\dagger\circ b, j^\dagger\circ c}\Big(\bigsqcup\nolimits_{i\in\mathbb{N}} j^\dagger \circ f_i\Big) \\ &= k^\dagger \circ \bigsqcup\nolimits_{i\in\mathbb{N}} \bar{\mathsf{W}}_{j^\dagger\circ b, j^\dagger\circ c}(j^\dagger \circ f_i) \\ &= \bigsqcup\nolimits_{i\in\mathbb{N}} k^\dagger \circ \bar{\mathsf{W}}_{j^\dagger\circ b, j^\dagger\circ c}(j \circ f_i) \\ &= \bigsqcup\nolimits_{i\in\mathbb{N}} \mathsf{W}_{b,c}(f_i). \end{aligned}$$

using the continuity of Kleisli compositions from Lemma 5.6.

□

Using the auxiliary functions, we define the denotational semantics of Clerical as in Figure 5.3.

In order to simplify the presentation, let us write $[\![e]\!]$ instead of $[\![\Gamma; \Delta \Vdash e : \tau]\!]$ or $[\![\Gamma \vdash e : \tau]\!]$ when it is obvious from the context what are missing.

Remark 5.1.

1. For a well-typed expression $\Gamma \vdash e : \tau$, its denotation on a state γ is a subset of $[\![\tau]\!] \cup \{\mathsf{e}, \bot\}$ of the values that e can evaluates to considering the nondeterminism in the evaluation. There are two special cases. The first is when $\bot \in [\![e]\!]\gamma$. The case denotes that there is a nondeterministic branch in the evaluation which leads to nontermination. The other case is when $\mathsf{e} \in [\![e]\!]\gamma$. It denotes the case where an error is occurred in the evaluation of e. See that due to the construction, in the case, $[\![e]\!]\gamma = \mathfrak{e}$.

The denotations of read-only expressions

$$\llbracket \Gamma \vdash c : \tau \rrbracket\, \gamma = \pi_1^\dagger \circ \llbracket \Gamma; \cdot \Vdash c : \tau \rrbracket\, \gamma\, ()$$

$$\llbracket x_1{:}\tau_1; \ldots, x_n{:}\tau_n \vdash x_i : \tau_i \rrbracket\, \gamma = \eta_{\llbracket \tau_i \rrbracket}\, \gamma_i$$

$$\llbracket \Gamma \vdash \texttt{false} : \mathsf{B} \rrbracket\, \gamma = \{ff\}$$

$$\llbracket \Gamma \vdash \texttt{true} : \mathsf{B} \rrbracket\, \gamma = \{tt\}$$

$$\llbracket \Gamma \vdash k : \mathsf{Z} \rrbracket\, \gamma = \{k\}$$

$$\llbracket \Gamma \vdash \texttt{skip} : \mathsf{U} \rrbracket\, \gamma = \{*\}$$

$$\llbracket \Gamma \vdash \iota(e) : \mathsf{R} \rrbracket\, \gamma = \iota^\dagger(\llbracket \Gamma \vdash e : \mathsf{Z} \rrbracket\, \gamma)$$

$$\llbracket \Gamma \vdash e_1 \odot e_2 : \mathsf{Z} \rrbracket\, \gamma = \llbracket \Gamma \vdash e_1 : \mathsf{Z} \rrbracket \gamma \odot^\dagger \llbracket \Gamma \vdash e_2 : \mathsf{Z} \rrbracket \gamma$$

$$\llbracket \Gamma \vdash e_1 \boxdot e_2 : \mathsf{R} \rrbracket\, \gamma = \llbracket \Gamma \vdash e_1 : \mathsf{R} \rrbracket \gamma \boxdot^\dagger \llbracket \Gamma \vdash e_2 : \mathsf{R} \rrbracket \gamma$$

$$\llbracket \Gamma \vdash e^{-1} : \mathsf{R} \rrbracket\, \gamma = (\llbracket \Gamma \vdash e : \mathsf{R} \rrbracket \gamma)^{(-1\downarrow_\perp)^{\ddagger\dagger}}$$

$$\llbracket \Gamma \vdash e_1 = e_2 : \mathsf{B} \rrbracket\, \gamma = \llbracket \Gamma \vdash e_1 : \mathsf{Z} \rrbracket \gamma =^\dagger \llbracket \Gamma \vdash e_2 : \mathsf{Z} \rrbracket \gamma$$

$$\llbracket \Gamma \vdash e_1 < e_2 : \mathsf{Z} \rrbracket\, \gamma = \llbracket \Gamma \vdash e_1 : \mathsf{Z} \rrbracket \gamma <^\dagger \llbracket \Gamma \vdash e_2 : \mathsf{Z} \rrbracket \gamma$$

$$\llbracket \Gamma \vdash e_1 \mathbin{\hat{<}} e_2 : \mathsf{R} \rrbracket\, \gamma = \llbracket \Gamma \vdash e_1 : \mathsf{R} \rrbracket \gamma (< \downarrow_\perp)^{\ddagger\dagger} \llbracket \Gamma \vdash e_2 : \mathsf{R} \rrbracket \gamma$$

$$\llbracket \Gamma \vdash (\texttt{lim}\ x.\ e) : \mathsf{R} \rrbracket\, \gamma = \mathsf{E}_{\mathbb{R}} \circ (\mathsf{lim}\downarrow_{\mathsf{e}})^{\ddagger\dagger}(\mathbb{N} \ni k \mapsto \llbracket \Gamma, x{:}\mathsf{Z} \vdash e : \mathsf{R} \rrbracket\, (\gamma, (x \mapsto k))).$$

The denotations of read-write expressions

$$\llbracket \Gamma; \Delta \Vdash e : \tau \rrbracket\, \gamma\, \delta = \mathsf{let}\ v \leftarrow \llbracket \Gamma, \Delta \vdash e : \tau \rrbracket\, (\gamma, \delta)\ \mathsf{in}\ (\delta, v)$$

$$\llbracket \Gamma; \Delta \Vdash c_1; c_2 : \tau \rrbracket\, \gamma\, \delta = \mathsf{let}\ (*, \delta') \leftarrow \llbracket \Gamma; \Delta \Vdash c_1 : \mathsf{U} \rrbracket\, \gamma\, \delta\ \mathsf{in}\ \llbracket \Gamma; \Delta \Vdash c_2 : \tau \rrbracket\, \gamma\, \delta'$$

$$\begin{aligned} \llbracket \Gamma; \Delta \Vdash (\texttt{var}\ x := e\ \texttt{in}\ c) : \tau \rrbracket\, \gamma\, \delta = {} & \mathsf{let}\ v \leftarrow \llbracket \Gamma, \Delta \vdash e : \sigma \rrbracket\, (\gamma, \delta)\ \mathsf{in} \\ & \mathsf{let}\ (\delta', v') \leftarrow \llbracket \Gamma; \Delta, x{:}\sigma \Vdash c : \tau \rrbracket\, \gamma\, (\delta, (x \mapsto v))\ \mathsf{in} \\ & (\delta'\ \restriction_{\mathrm{dom}(\Delta)}, v') \end{aligned}$$

$$\llbracket \Gamma; \Delta \Vdash (x := e) : \mathsf{U} \rrbracket\, \gamma\, \delta = \mathsf{let}\ v \leftarrow \llbracket \Gamma, \Delta \vdash e : \tau \rrbracket\, (\gamma, \delta)\ \mathsf{in}\ \eta_{\llbracket \Delta \rrbracket \times \mathbb{1}}(\delta[x \mapsto v], *)$$

$$\begin{aligned} \llbracket \Gamma; \Delta \Vdash (\texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{end}) : \tau \rrbracket\, \gamma\, \delta = {} & \mathsf{let}\ b \leftarrow \llbracket \Gamma, \Delta \vdash e : \mathsf{U} \rrbracket\, (\gamma, \delta)\ \mathsf{in} \\ & \mathsf{Cond}_{\mathbb{P}_\star(\llbracket \Delta \rrbracket \times \llbracket \tau \rrbracket)}(b, \llbracket \Gamma; \Delta \Vdash c_1 : \tau \rrbracket\, \gamma\, \delta, \llbracket \Gamma; \Delta \Vdash c_2 : \tau \rrbracket\, \gamma\, \delta) \end{aligned}$$

The denotation of guarded nondeterminism

$$\begin{aligned} & \llbracket \Gamma; \Delta \Vdash (\texttt{case}\ e_1 \Rightarrow c_1 \mid \cdots \mid e_n \Rightarrow c_n\ \texttt{end}) : \tau \rrbracket\, \gamma\, \delta = \\ & \qquad \left(\bigcup_{i=1}^{n} \bigcup_{b \in \llbracket \Gamma, \Delta \vdash e_i : \mathsf{B} \rrbracket\, (\gamma, \delta)} \mathsf{Guard}_{\llbracket \tau \rrbracket}(b, \llbracket \Gamma; \Delta \Vdash c_i : \tau \rrbracket\, \gamma\, \delta) \right)_\star \end{aligned}$$

The denotation of a loop

$$\llbracket \Gamma; \Delta \Vdash (\texttt{while}\ e\ \texttt{do}\ c\ \texttt{end}) : \mathsf{U} \rrbracket\, \gamma = \mathsf{LFP}(\mathsf{W}_{\llbracket \Gamma, \Delta \vdash b : \mathsf{B} \rrbracket|_\gamma, \llbracket \Gamma; \Delta \Vdash c : \mathsf{U} \rrbracket \gamma})$$

Figure 5.3: The denotational semantics of Clerical.

2. The two failures, nontermination $\bot$ and error $\mathtt{e}$, are strictly distinguished in the semantics. It is because nontermination can be meaningful when it is an argument of our guarded nondeterminism.[3] The source of $\bot$ is (i) division by 0, (ii) comparing an identical real number, and (iii) infinite while loop. On the other hand, the source of $\mathtt{e}$ is an ill-defined limit.

3. For a well-typed case expression $\Gamma; \Delta \Vdash \mathtt{case}\ e_1 \Rightarrow c_1 \mid e_2 \Rightarrow c_2\ \mathtt{end} : \tau$, even when e_1 is nontermination, if e_2 evaluates to tt, c_2 executes. When both e_1 and e_2 evaluate to tt, the denotation of the case expression contains the both branches. When both e_1 and e_2 evaluate to ff, the denotation of the case expression contains $\bot$.

4. The denotation of a while loop is defined in the way that it satisfies the recurrence equation:

$$[\![\mathtt{while}\ e\ \mathtt{do}\ c\ \mathtt{end}]\!] = [\![\mathtt{if}\ e\ \mathtt{then}\ c; \mathtt{while}\ e\ \mathtt{do}\ c\ \mathtt{end}\ \mathtt{else}\ \mathtt{skip}\ \mathtt{end}]\!]$$

5. Define a sequence of expressions:

$$A^{n+1}_{e,c} \coloneqq \mathtt{if}\ e\ \mathtt{then}\ c; A^n_{e,c}\ \mathtt{else}\ \mathtt{skip}\ \mathtt{end}$$
$$A^0_{e,c} \coloneqq \mathtt{while}\ \mathtt{true}\ \mathtt{do}\ \mathtt{skip}\ \mathtt{end}.$$

   Then, the sequence of denotations $\left([\![A^n_{e,c}]\!]\right)_{n\in\mathbb{N}}$ forms a chain whose limit is the denotation of the while loop.

6. For a well-typed expression $c' \coloneqq \mathtt{while}\ e\ \mathtt{do}\ c\ \mathtt{end}$ and a state $(\gamma; \delta)$, the following holds:

$$[\![c']\!]\gamma\ \delta = \mathfrak{e} \text{ if and only if there is } m \in \mathbb{N} \text{ s.t. } [\![A^m_{e,c}]\!]\gamma\ \delta = \mathfrak{e}$$

   If it is not the case, $(\delta', *) \in [\![c']\!]\gamma\ \delta$ holds if and only if there is $n \in \mathbb{N}$ such that $(\delta', *) \in [\![A^n_{e,c}]\!]\gamma\ \delta$.

5.4 Reasoning Principles

5.4.1 Assertion Language

We consider a many-sorted first order logic over the sorts $\mathsf{U}, \mathsf{B}, \mathsf{Z}, \mathsf{R}$. Its term language includes the subset of read-only expressions of Clerical by taking (1) constants $\mathtt{true}, \mathtt{false}, \mathtt{skip}, k$, (2) arithmetical operations $e_1 \odot e_2$, $e_2 \boxdot e_2$, $e^{-\mathbf{1}}$, (3) coercion $\iota(e)$, and (4) variables. We let its well-typedness be inherited from the well-typedness of Clerical expressions. Furthermore, we suppose $\mathbb{Z} \ni p \mapsto 2^p \in \mathbb{R}$ is expressible in the logical language that there is a typing rule

$$\frac{\Gamma \vdash p : \mathsf{Z}}{\Gamma \vdash 2^p : \mathsf{R}}$$

For a context Γ, we define the well-formed judgement of formulae:

$$\frac{}{\Gamma \Vdash \mathsf{True}} \qquad \frac{}{\Gamma \Vdash \mathsf{False}} \qquad \frac{\Gamma \Vdash t_1 : \tau \qquad \Gamma \Vdash t_2 : \tau}{\Gamma \Vdash t_1 = t_2} \qquad \frac{\Gamma \Vdash t_1 : \mathsf{Z} \qquad \Gamma \Vdash t_2 : \mathsf{Z}}{\Gamma \Vdash t_1 < t_2} \qquad \frac{\Gamma \Vdash t_1 : \mathsf{R} \qquad \Gamma \Vdash t_2 : \mathsf{R}}{\Gamma \Vdash t_1 < t_2}$$

$$\frac{\Gamma \Vdash \phi \qquad \Gamma \Vdash \psi}{\Gamma \Vdash \phi \Rightarrow \psi} \qquad \frac{\Gamma \Vdash \phi \qquad \Gamma \Vdash \psi}{\Gamma \Vdash \phi \wedge \psi} \qquad \frac{\Gamma \Vdash \phi \qquad \Gamma \Vdash \psi}{\Gamma \Vdash \phi \vee \psi} \qquad \frac{\Gamma, x : \tau \Vdash \psi}{\Gamma \Vdash \exists x : \tau\,.\,\psi} \qquad \frac{\Gamma, x : \tau \Vdash \psi}{\Gamma \Vdash \forall x : \tau\,.\,\psi}$$

Familiar formulae are considered to be defined as abbreviations. For example, $\neg\phi$ is an abbreviation for $\phi \Rightarrow \mathsf{False}$, $x \leq y$ is an abbreviation for $x = y \vee x < y$, and so on.

[3] In ERC from Chapter 3, we identified both with $\bot$.

We take the standard interpretation and write $[\![\Gamma \Vdash \phi]\!]$ for the set of states $\gamma \in [\![\Gamma]\!]$ that validate ϕ. Though we do not specify derivation rules of the assertion language, let us write $\Gamma \vdash \phi$ to denote that the well-formed formula $\Gamma \Vdash \phi$ is derivable in the language. Of course, we assume the assertion language is sound.

5.4.2 Specifications

We use precondition-postcondition-style program specification. Given a well-typed read-only expression $\Gamma \vdash e : \tau$, we say a context Ξ to be *a context of auxiliary variables* if its domain is disjoint to Γ. Similarly, for a well-typed read-write expression $\Gamma; \Delta \Vdash e : \tau$, we say a context Ξ to be a context of auxiliary variables if its domain is disjoint to Γ, Δ.

We define the four kinds of specifications.

1. *Partial correctness specification for read-only expressions* For a well-typed read-only expression $\Gamma \vdash e : \tau$, a context of auxiliary variables Ξ, a precondition $\Xi, \Gamma \Vdash \phi$, and a postcondition $\Xi, \Gamma, y{:}\tau \Vdash \psi$,

$$\Xi \triangleright \Gamma \vdash \{\phi\}\ \ e\ \ \{y : \tau \mid \psi\}$$

denotes that

$$\forall (\xi, \gamma) \in [\![\Xi, \Gamma \Vdash \phi]\!] \,.\, \mathsf{e} \notin [\![\Gamma \vdash e : \tau]\!]\, \gamma \wedge \forall v \in [\![\Gamma \vdash e : \tau]\!]\, \gamma \,.\, (\xi, \gamma, y \mapsto v) \in [\![\Xi, \Gamma, y{:}\tau \Vdash \psi]\!] \,.$$

2. *Partial correctness specification for read-write expressions* For a well-typed read-write expression $\Gamma; \Delta \Vdash e : \tau$, a context of auxiliary variables Ξ, a precondition $\Xi, \Gamma, \Delta \Vdash \phi$, and a postcondition $\Xi, \Gamma, \Delta, y{:}\tau \Vdash \psi$,

$$\Xi \triangleright \Gamma; \Delta \Vdash \{\phi\}\ \ e\ \ \{y : \tau \mid \psi\}$$

denotes that

$$\begin{aligned} &\forall (\xi, \gamma, \delta) \in [\![\Xi, \Gamma, \Delta \Vdash \phi]\!] \,. \\ &\quad \mathsf{e} \notin [\![\Gamma \vdash e : \tau]\!]\, \gamma\, \delta \wedge \forall (\delta', v) \in [\![\Gamma \vdash e : \tau]\!]\, \gamma\, \delta \,.\, (\xi, \gamma, \delta', y \mapsto v) \in [\![\Xi, \Gamma, \Delta, y{:}\tau \Vdash \psi]\!] \,. \end{aligned}$$

3. *Total correctness specification for read-only expressions* For a well-typed read-only expression $\Gamma \vdash e : \tau$, a context of auxiliary variables Ξ, a precondition $\Xi, \Gamma \Vdash \phi$, and a postcondition $\Xi, \Gamma, y{:}\tau \Vdash \psi$,

$$\Xi \triangleright \Gamma \vdash \{\phi\}\ \ e\ \ \downarrow\{y : \tau \mid \psi\}$$

denotes that

$$\forall (\xi, \gamma) \in [\![\Xi, \Gamma \Vdash \phi]\!] \,.\, \bot \notin [\![\Gamma \vdash e : \tau]\!]\, \gamma \wedge \forall v \in [\![\Gamma \vdash e : \tau]\!]\, \gamma \,.\, (\xi, \gamma, y \mapsto v) \in [\![\Xi, \Gamma, y{:}\tau \Vdash \psi]\!] \,.$$

4. *Total correctness specification for read-write expressions* For a well-typed read-write expression $\Gamma; \Delta \Vdash e : \tau$, a context of auxiliary variables Ξ, a precondition $\Xi, \Gamma, \Delta \Vdash \phi$, and a postcondition $\Xi, \Gamma, \Delta, y{:}\tau \Vdash \psi$,

$$\Xi \triangleright \Gamma \Vdash \{\phi\}\ \ e\ \ \downarrow\{y : \tau \mid \psi\}$$

denotes that

$$\begin{aligned} &\forall (\xi, \gamma, \delta) \in [\![\Xi, \Gamma, \Delta \Vdash \phi]\!] \,. \\ &\quad \bot \notin [\![\Gamma \vdash e : \tau]\!]\, \gamma\, \delta \wedge \forall (\delta', v) \in [\![\Gamma \vdash e : \tau]\!]\, \gamma\, \delta \,.\, (\xi, \gamma, \delta', y \mapsto v) \in [\![\Xi, \Gamma, \Delta, y{:}\tau \Vdash \psi]\!] \,. \end{aligned}$$

5.4.3 Proof Rules

We define a formal system for deriving correctness specifications. The first rule is for deriving partial correctness specifications from total correctness specifications.

•*Rules for partial correctness from total correctness*

$$\frac{\Xi \triangleright \Gamma \vdash \{\phi\}\ c\ \downarrow\{y:\tau \mid \psi\}}{\Xi \triangleright \Gamma \vdash \{\phi\}\ c\ \{y:\tau \mid \psi\}} \qquad \frac{\Xi \triangleright \Gamma; \Delta \Vdash \{\phi\}\ c\ \downarrow\{y:\tau \mid \psi\}}{\Xi \triangleright \Gamma; \Delta \Vdash \{\phi\}\ c\ \{y:\tau \mid \psi\}}$$

In order to simplify the presentation, when there are rules in a similar form where the only difference is that one is for a partial correctness specification and the other is for a total correctness specification, we write it as one rule with putting '?' in the place of '↓'. Replacing every occurrence of ? with ↓ and with the blank gives us two rules, though it is written only once.

•*Rules for read-only-read-write*

$$\frac{\Xi \triangleright \Gamma; \cdot \Vdash \{\phi\}\ c\ ?\{y:\tau \mid \psi\}}{\Xi \triangleright \Gamma \vdash \{\phi\}\ c\ ?\{y:\tau \mid \psi\}} \qquad \frac{\Xi \triangleright \Gamma, \Delta \vdash \{\phi\}\ e\ ?\{y:\tau \mid \psi\}}{\Xi \triangleright \Gamma; \Delta \Vdash \{\phi\}\ e\ ?\{y:\tau \mid \psi\}}$$

•*Rules for extending context*

$$\frac{\Xi \subseteq \Xi' \quad \Gamma \subseteq \Gamma' \qquad \Xi \triangleright \Gamma \vdash \{\phi\}\ c\ ?\{y:\tau \mid \psi\}}{\Xi' \triangleright \Gamma' \vdash \{\phi\}\ c\ ?\{y:\tau \mid \psi\}} \qquad \frac{\Xi \subseteq \Xi' \quad \Gamma \subseteq \Gamma' \quad \Delta \subseteq \Delta' \qquad \Xi \triangleright \Gamma; \Delta \Vdash \{\phi\}\ c\ ?\{y:\tau \mid \psi\}}{\Xi' \triangleright \Gamma'; \Delta' \Vdash \{\phi\}\ c\ ?\{y:\tau \mid \psi\}}$$

Here, for two typing contexts Γ and Δ, $\Gamma \subseteq \Delta$ denotes that $\Gamma \vdash x : \tau \Rightarrow \Delta \vdash x : \tau$ for any variable x and data type τ.

•*Rules for read-only variables*

$$\frac{\Xi, \Gamma \Vdash \theta \qquad \Xi \triangleright \Gamma \vdash \{\phi\}\ c\ ?\{y:\tau \mid \psi\}}{\Xi' \triangleright \Gamma \vdash \{\phi \wedge \theta\}\ c\ ?\{y:\tau \mid \psi \wedge \theta\}} \qquad \frac{\Xi, \Gamma \Vdash \theta \qquad \Xi \triangleright \Gamma \vdash \{\phi \vee \theta\}\ c\ ?\{y:\tau \mid \mathsf{True}\} \qquad \Xi \triangleright \Gamma \vdash \{\phi\}\ c\ ?\{y:\tau \mid \psi\}}{\Xi' \triangleright \Gamma \Vdash \{\phi \vee \theta\}\ c\ ?\{y:\tau \mid \psi \vee \theta\}}$$

•*Rules for precondition strengthening and postcondition weakening*

$$\frac{\Xi, \Gamma \vdash \phi \Rightarrow \phi' \quad \Xi, \Gamma, y{:}\tau \vdash \psi' \Rightarrow \phi \qquad \Xi \triangleright \Gamma \vdash \{\phi'\}\ c\ ?\{y:\tau \mid \psi'\}}{\Xi \triangleright \Gamma \vdash \{\phi\}\ c\ ?\{y:\tau \mid \psi\}} \qquad \frac{\Xi, \Gamma \vdash \phi \Rightarrow \phi' \quad \Xi, \Gamma, \Delta, y{:}\tau \vdash \psi' \Rightarrow \phi \qquad \Xi \triangleright \Gamma; \Delta \Vdash \{\phi'\}\ c\ ?\{y:\tau \mid \psi'\}}{\Xi \triangleright \Gamma; \Delta \Vdash \{\phi\}\ c\ ?\{y:\tau \mid \psi\}}$$

•*Rules for conjunctions of assertions*

$$\frac{\Xi \triangleright \Gamma \vdash \{\phi_1\}\ c\ ?\{y:\tau \mid \psi_1\} \qquad \Xi \triangleright \Gamma \vdash \{\phi_2\}\ c\ ?\{y:\tau \mid \psi_2\}}{\Xi \triangleright \Gamma \vdash \{\phi_1 \wedge \phi_2\}\ c\ ?\{y:\tau \mid \psi_1 \wedge \psi_2\}} \qquad \frac{\Xi \triangleright \Gamma; \Delta \Vdash \{\phi_1\}\ c\ ?\{y:\tau \mid \psi_1\} \qquad \Xi \triangleright \Gamma; \Delta \Vdash \{\phi_2\}\ c\ ?\{y:\tau \mid \psi_2\}}{\Xi \triangleright \Gamma; \Delta \Vdash \{\phi_1 \wedge \phi_2\}\ c\ ?\{y:\tau \mid \psi_1 \wedge \psi_2\}}$$

•*Rules for disjunctions of assertions*

$$\frac{\Xi \triangleright \Gamma \vdash \{\phi_1\}\ c\ ?\{y:\tau \mid \psi_1\} \qquad \Xi \triangleright \Gamma \vdash \{\phi_2\}\ c\ ?\{y:\tau \mid \psi_2\}}{\Xi \triangleright \Gamma \vdash \{\phi_1 \vee \phi_2\}\ c\ ?\{y:\tau \mid \psi_1 \vee \psi_2\}} \qquad \frac{\Xi \triangleright \Gamma; \Delta \Vdash \{\phi_1\}\ c\ ?\{y:\tau \mid \psi_1\} \qquad \Xi \triangleright \Gamma; \Delta \Vdash \{\phi_1\}\ c\ ?\{y:\tau \mid \psi_1\}}{\Xi \triangleright \Gamma; \Delta \Vdash \{\phi_1 \vee \phi_2\}\ c\ ?\{y:\tau \mid \psi_1 \vee \psi_2\}}$$

The proof rules for variables and constants form axioms.

•*Rule for variable*

$$\frac{}{\Xi \triangleright x_1{:}\tau_1, \cdots, x_n{:}\tau_n \vdash \{\theta(x_i)\}\ \ x_i\ \ \downarrow\{y : \tau_n \mid \theta(y)\}}$$

•*Rules for constants*

$$\frac{}{\Xi \triangleright \Gamma \vdash \{\theta[\mathtt{true}/y]\}\ \ \mathtt{true}\ \ \downarrow\{y : \mathsf{B} \mid \theta\}} \qquad \frac{}{\Xi \triangleright \Gamma \vdash \{\theta[\mathtt{false}/y]\}\ \ \mathtt{false}\ \ \downarrow\{y : \mathsf{B} \mid \theta\}}$$

$$\frac{}{\Xi \triangleright \Gamma \vdash \{\theta[k/x]\}\ \ k\ \ \downarrow\{y : \mathsf{Z} \mid \theta\}} \qquad \frac{}{\Xi \triangleright \Gamma \vdash \{\theta[\mathtt{skip}/x]\}\ \ \mathtt{skip}\ \ \downarrow\{y : \mathsf{U} \mid \theta\}}$$

•*Rule for coercion from* Z *to* R

$$\frac{\Xi \triangleright \Gamma \vdash \{\phi\}\ \ e\ \ ?\{y : \mathsf{Z} \mid \theta\}}{\Xi \triangleright \Gamma \vdash \{\phi\}\ \ \iota(e)\ \ ?\{z : \mathsf{R} \mid \exists y{:}\mathsf{R}\,.\,\theta \wedge z = \iota(y)\}}$$

•*Rules for integer arithmetic* $\odot \in \{+, -, \times\}$

$$\frac{\begin{array}{l}\Xi \triangleright \Gamma \vdash \{\phi\}\ \ e_1\ \ ?\{y : \mathsf{Z} \mid \mathsf{True}\} \quad \Xi, x_1{:}\mathsf{Z} \triangleright \Gamma \vdash \{\phi_1\}\ \ e_1\ \ \{y : \mathsf{Z} \mid y \neq x_1\} \\ \Xi \triangleright \Gamma \vdash \{\phi\}\ \ e_2\ \ ?\{y : \mathsf{Z} \mid \mathsf{True}\} \quad \Xi, x_2{:}\mathsf{Z} \triangleright \Gamma \vdash \{\phi_2\}\ \ e_1\ \ \{y : \mathsf{Z} \mid y \neq x_2\}\end{array}}{\Xi \triangleright \Gamma \vdash \{\phi\}\ \ e_1 \odot e_2\ \ ?\{y : \mathsf{R} \mid \theta\}}\ \begin{array}{l}\forall x_1, x_2{:}\mathsf{Z}\,.\,\phi \wedge \neg\phi_1 \wedge \neg\phi_2 \\ \Rightarrow \psi[(x_1 \odot x_2)/y]\end{array}$$

•*Rules for real arithmetic* $\boxdot \in \{+, -, \times\}$

$$\frac{\begin{array}{l}\Xi \triangleright \Gamma \vdash \{\phi\}\ \ e_1\ \ ?\{y : \mathsf{R} \mid \mathsf{True}\} \quad \Xi, x_1{:}\mathsf{R} \triangleright \Gamma \vdash \{\phi_1\}\ \ e_1\ \ \{y : \mathsf{R} \mid y \neq x_1\} \\ \Xi \triangleright \Gamma \vdash \{\phi\}\ \ e_2\ \ ?\{y : \mathsf{R} \mid \mathsf{True}\} \quad \Xi, x_2{:}\mathsf{R} \triangleright \Gamma \vdash \{\phi_2\}\ \ e_1\ \ \{y : \mathsf{R} \mid y \neq x_2\}\end{array}}{\Xi \triangleright \Gamma \vdash \{\phi\}\ \ e_1 \boxdot e_2\ \ ?\{y : \mathsf{R} \mid \psi\}}\ \begin{array}{l}\forall x_1, x_2{:}\mathsf{R}\,.\,\phi \wedge \neg\phi_1 \wedge \neg\phi_2 \\ \Rightarrow \psi[(x_1 \boxdot x_2)/y]\end{array}$$

In the rules of real or integer arithmetic, the first premise $\Xi \triangleright \Gamma, \Delta \vdash \{\phi\}\ \ e_i\ \ ?\{y : \tau \mid \mathsf{True}\}$ ensures that when a state is in ϕ, the evaluations of e_1 and e_2 are well-defined and terminate in the case $? = \downarrow$. The second premise $\Xi; x_i{:}\tau \triangleright \Gamma, \Delta \vdash \{\phi_i\}\ \ e_i\ \ \{y : \tau \mid y \neq x_i\}$ uses an auxiliary variable x_i to represent the value of e_i. It ensures that any state that makes the evaluation of e_i contain a nondeterministic branch which results x_i is in $\neg\phi_i$. And, the side-condition ensures that for any state in ϕ, if it makes e_1 result in x_1 and e_2 result in x_2, the state is in $\psi[(x_1 \odot x_2)/y]$ or in $\psi[(x_1 \boxdot x_2)/y]$.

Integer comparisons can be similarly dealt with.

•*Rules for integer comparisons*

$$\frac{\begin{array}{l}\Xi \triangleright \Gamma \vdash \{\phi\}\ \ e_1\ \ ?\{y : \mathsf{Z} \mid \mathsf{True}\} \quad \Xi, x_1{:}\mathsf{Z} \triangleright \Gamma \vdash \{\phi_1\}\ \ e_1\ \ \{y : \mathsf{Z} \mid y \neq x_1\} \\ \Xi \triangleright \Gamma \vdash \{\phi\}\ \ e_2\ \ ?\{y : \mathsf{Z} \mid \mathsf{True}\} \quad \Xi, x_2{:}\mathsf{Z} \triangleright \Gamma \vdash \{\phi_2\}\ \ e_1\ \ \{y : \mathsf{Z} \mid y \neq x_2\}\end{array}}{\Xi \triangleright \Gamma \vdash \{\phi\}\ \ e_1 < e_2\ \ ?\{y : \mathsf{Z} \mid \psi\}}\ \begin{array}{l}\forall x_1, x_2{:}\mathsf{Z}\,.\,\phi \wedge \neg\phi_1 \wedge \neg\phi_2 \\ \Rightarrow \big((x_1 < x_2 \Rightarrow \psi[\mathtt{true}/y]) \\ \quad \wedge (x_1 \geq x_2 \Rightarrow \psi[\mathtt{false}/y])\big)\end{array}$$

$$\frac{\begin{array}{l}\Xi \triangleright \Gamma \vdash \{\phi\}\ \ e_1\ \ ?\{y : \mathsf{Z} \mid \mathsf{True}\} \quad \Xi, x_1{:}\mathsf{Z} \triangleright \Gamma \vdash \{\phi_1\}\ \ e_1\ \ \{y : \mathsf{Z} \mid y \neq x_1\} \\ \Xi \triangleright \Gamma \vdash \{\phi\}\ \ e_2\ \ ?\{y : \mathsf{Z} \mid \mathsf{True}\} \quad \Xi, x_2{:}\mathsf{Z} \triangleright \Gamma \vdash \{\phi_2\}\ \ e_1\ \ \{y : \mathsf{Z} \mid y \neq x_2\}\end{array}}{\Xi \triangleright \Gamma \vdash \{\phi\}\ \ e_1 = e_2\ \ ?\{y : \mathsf{Z} \mid \psi\}}\ \begin{array}{l}\forall x_1, x_2{:}\mathsf{Z}\,.\,\phi \wedge \neg\phi_1 \wedge \neg\phi_2 \\ \Rightarrow \big((x_1 = x_2 \Rightarrow \psi[\mathtt{true}/y]) \\ \quad \wedge (x_1 \neq x_2 \Rightarrow \psi[\mathtt{false}/y])\big)\end{array}$$

Total correctness regarding the partial operations $<$ and $^{-1}$ are more tricky. We should ensure that the input values of the operators that can make the result diverge are excluded using side-conditions.

•*Rules for real number comparison*

$$\frac{\begin{array}{l}\Xi \triangleright \Gamma \vdash \{\phi\}\ e_1\ \{y : \mathsf{R} \mid \mathsf{True}\} \quad \Xi, x_1{:}\mathsf{R} \triangleright \Gamma \vdash \{\phi_1\}\ e_1\ \{y : \mathsf{R} \mid y \neq x_1\} \\ \Xi \triangleright \Gamma \vdash \{\phi\}\ e_2\ \{y : \mathsf{R} \mid \mathsf{True}\} \quad \Xi, x_2{:}\mathsf{R} \triangleright \Gamma \vdash \{\phi_2\}\ e_1\ \{y : \mathsf{R} \mid y \neq x_2\}\end{array}}{\Xi \triangleright \Gamma \vdash \{\phi\}\ e_1 < e_2\ \{y : \mathsf{B} \mid \psi\}} \quad \begin{array}{l}\forall x_1, x_2{:}\mathsf{R}\,.\,\phi \wedge \neg\phi_1 \wedge \neg\phi_2 \\ \Rightarrow \big((x_1 < x_2 \Rightarrow \psi[\mathtt{true}/y]) \\ \quad \wedge (x_1 > x_2 \Rightarrow \psi[\mathtt{false}/y])\big)\end{array}$$

$$\frac{\begin{array}{l}\Xi \triangleright \Gamma \vdash \{\phi\}\ e_1\ \downarrow\{y : \mathsf{R} \mid \mathsf{True}\} \quad \Xi, x_1{:}\mathsf{R} \triangleright \Gamma \vdash \{\phi_1\}\ e_1\ \{y : \mathsf{R} \mid y \neq x_1\} \\ \Xi \triangleright \Gamma \vdash \{\phi\}\ e_2\ \downarrow\{y : \mathsf{R} \mid \mathsf{True}\} \quad \Xi, x_2{:}\mathsf{R} \triangleright \Gamma \vdash \{\phi_2\}\ e_1\ \{y : \mathsf{R} \mid y \neq x_2\}\end{array}}{\Xi \triangleright \Gamma \vdash \{\phi\}\ e_1 < e_2\ \downarrow\{y : \mathsf{B} \mid \psi\}} \quad \begin{array}{l}\forall x_1, x_2{:}\mathsf{R}\,.\,\phi \wedge \neg\phi_1 \wedge \neg\phi_2 \\ \Rightarrow \big((x_1 < x_2 \Rightarrow \psi[\mathtt{true}/y]) \\ \quad \wedge (x_1 > x_2 \Rightarrow \psi[\mathtt{false}/y])\big) \\ \quad \wedge x_1 \neq x_2\end{array}$$

•*Rules for multiplicative inversion*

$$\frac{\Xi \triangleright \Gamma \vdash \{\phi\}\ e\ \{y : \mathsf{Z} \mid \mathsf{True}\} \quad \Xi, x{:}\mathsf{Z} \triangleright \Gamma \vdash \{\phi'\}\ e\ \{y : \mathsf{R} \mid y \neq x\}}{\Xi \triangleright \Gamma \vdash \{\phi\}\ e^{-1}\ \{y : \mathsf{R} \mid \psi\}} \quad \begin{array}{l}\forall x{:}\mathsf{R}\,.\,\phi \wedge \neg\phi' \\ \Rightarrow \wedge\psi[x^{-1}/y]\end{array}$$

$$\frac{\Xi \triangleright \Gamma \vdash \{\phi\}\ e\ \downarrow\{y : \mathsf{Z} \mid \mathsf{True}\} \quad \Xi, x{:}\mathsf{Z} \triangleright \Gamma \vdash \{\phi'\}\ e\ \{y : \mathsf{R} \mid y \neq x\}}{\Xi \triangleright \Gamma \vdash \{\phi\}\ e^{-1}\ \downarrow\{y : \mathsf{R} \mid \psi\}} \quad \begin{array}{l}\forall x{:}\mathsf{R}\,.\,\phi \wedge \neg\phi' \\ \Rightarrow x \neq 0 \wedge \psi[x^{-1}/y]\end{array}$$

•*Rule for limit*

$$\frac{\Xi, z : \mathsf{R} \triangleright \Gamma, x : \mathsf{Z} \vdash \{\phi'\}\ e\ \downarrow\{y : \mathsf{R} \mid \psi'\}}{\Xi \triangleright \Gamma \vdash \{\phi\}\ \mathtt{lim}\, x\,.\,e\ \downarrow\{z : \mathsf{R} \mid \psi\}} \quad \begin{array}{l}\phi \Rightarrow \\ \exists z{:}\mathsf{R}\,.\,(\forall x{:}\mathsf{Z}\,.\,x \geq 0 \Rightarrow \phi' \wedge (\forall y.\ \psi' \Rightarrow |y - z| \leq 2^{-x})) \wedge \psi\end{array}$$

In the rule for limit, the premise $\Xi, z{:}\mathsf{R} \triangleright \Gamma, x{:}\mathsf{Z} \vdash \{\phi'\}\ e\ \downarrow\{y : \mathsf{R} \mid \psi'\}$ uses an auxiliary variable $z : \mathsf{R}$ to represent the limit. The premise ensures that any states in ϕ' makes the evaluation of e be well-defined, terminate, and be in ψ'.

Suppose a state that satisfies the precondition ϕ. Then, the side-condition ensures that there is a real number z where for any natural number x, the precondition of the premise ϕ' holds. The premise then says the evaluation of e results y, which is in ψ'. And, the premise says such y is 2^{-x} approximation of z.

Since the side condition ensures such z in ψ, the evaluation of the limit expression, which is z, is in ψ.

•*Rule for sequencing*

$$\frac{\Xi \triangleright \Gamma; \Delta \Vdash \{\phi\}\ c_1\ ?\{y : \mathsf{U} \mid \psi\} \quad \Xi \triangleright \Gamma; \Delta \Vdash \{\psi[\mathtt{skip}/y]\}\ c_2\ ?\{y : \tau \mid \theta\}}{\Xi \triangleright \Gamma; \Delta \Vdash \{\phi\}\ c_1; c_2\ ?\{y : \tau \mid \theta\}}$$

•*Rule for local variable*

$$\frac{\begin{array}{c}\Xi \triangleright \Gamma, \Delta \vdash \{\phi\}\ e\ ?\{x : \sigma \mid \psi\} \\ \Xi \triangleright x{:}\sigma, \Gamma; \Delta \Vdash \{\psi\}\ c\ ?\{y : \tau \mid \theta\}\end{array}}{\Xi \triangleright \Gamma; \Delta \Vdash \{\phi\}\ \mathtt{var}\ x := e\ \mathtt{in}\ c\ ?\{y : \tau \mid \exists x{:}\sigma\,.\,\theta\}}$$

•*Rule for assignment*

$$\frac{\Xi \triangleright \Gamma, \Delta \vdash \{\phi\}\ e\ ?\{y : \tau \mid \psi\}}{\Xi \triangleright \Gamma; \Delta \Vdash \{\phi \wedge \forall y : \tau\,.\,(\psi \Rightarrow \theta[y/x])\}\ x := e\ ?\{\theta\}}$$

•*Rule for conditional*

$$\frac{\begin{array}{l}\Xi \triangleright \Gamma, \Delta \vdash \{\phi\}\ e\ ?\{y : \mathsf{B} \mid \mathsf{True}\} \\ \Xi \triangleright \Gamma, \Delta \vdash \{\phi_{\text{false}}\}\ e\ \{y : \mathsf{B} \mid y = \texttt{false}\} \quad \Xi \triangleright \Gamma; \Delta \Vdash \{\phi \wedge \neg\phi_{\text{false}}\}\ c_1\ ?\{y : \tau \mid \psi\} \\ \Xi \triangleright \Gamma, \Delta \vdash \{\phi_{\text{true}}\}\ e\ \{y : \mathsf{B} \mid y = \texttt{true}\} \quad \Xi \triangleright \Gamma; \Delta \Vdash \{\phi \wedge \neg\phi_{\text{true}}\}\ c_2\ ?\{y : \tau \mid \psi\}\end{array}}{\Xi \triangleright \Gamma; \Delta \Vdash \{\phi\}\ \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \texttt{end}\ ?\{y : \tau \mid \psi\}}$$

•*Partial correctness rule for guarded cases*

$$\frac{\begin{array}{l}\Xi \triangleright \Gamma, \Delta \vdash \{\phi\}\ e_i\ \{y : \mathsf{B} \mid \mathsf{True}\} \\ \Xi \triangleright \Gamma, \Delta \vdash \{\theta_i\}\ e_i\ \{y : \mathsf{B} \mid y = \texttt{false}\} \\ \Xi \triangleright \Gamma; \Delta \Vdash \{\phi \wedge \neg\theta_i\}\ c_i\ \{y : \tau \mid \psi\} \quad (i = 1, \cdots, n)\end{array}}{\Xi \triangleright \Gamma; \Delta \Vdash \{\phi\}\ \texttt{case}\ e_1 \Rightarrow c_1 \mid \cdots \mid e_n \Rightarrow c_n\ \texttt{end}\ \{y : \tau \mid \psi\}}$$

•*Partial correctness rule for guarded cases*

$$\frac{\begin{array}{l}\Xi \triangleright \Gamma, \Delta \vdash \{\phi\}\ e_i\ \{y : \mathsf{B} \mid \mathsf{True}\} \\ \Xi \triangleright \Gamma, \Delta \vdash \{\phi_i\}\ e_i\ \downarrow\{y : \mathsf{B} \mid y = \texttt{true}\} \\ \Xi \triangleright \Gamma, \Delta \vdash \{\theta_i\}\ e_i\ \{y : \mathsf{B} \mid y = \texttt{false}\} \\ \Xi \triangleright \Gamma; \Delta \Vdash \{\phi \wedge (\phi_1 \vee \cdots \vee \phi_n) \wedge \neg\theta_i\}\ c_i\ \downarrow\{y : \tau \mid \psi\} \quad (i = 1, \cdots, n)\end{array}}{\Xi \triangleright \Gamma; \Delta \Vdash \{\phi \wedge (\phi_1 \vee \cdots \vee \phi_n)\}\ \texttt{case}\ e_1 \Rightarrow c_1 \mid \cdots \mid e_n \Rightarrow c_n\ \texttt{end}\ \downarrow\{y : \tau \mid \psi\}}$$

The first premise $\Xi \triangleright \Gamma, \Delta \vdash \{\phi\}\ e_i\ \{y : \mathsf{B} \mid \mathsf{True}\}$ ensures that any state in ϕ makes the evaluation of e_1 and e_2 be well-defined. The second premise $\Xi \triangleright \Gamma, \Delta \vdash \{\phi_i\}\ e_i\ \downarrow\{y : \mathsf{B} \mid y = \texttt{true}\}$ ensures that states in $\phi_1 \vee \phi_2$ make either e_1 or e_2 evaluates *only* to `true`. Hence, any such state will not make the evaluation of the case expression nonterminate due to having a branch which does not satisfy any of the guards.

The third premise $\Xi \triangleright \Gamma, \Delta \vdash \{\theta_i\}\ e_i\ \{y : \mathsf{B} \mid y = \texttt{false}\}$ states that any state that may make e_i evaluate to `true` is in $\neg\theta_i$. Hence any state in $\phi \wedge (\phi_1 \vee \phi_2)$ that may make e_i yield `true` is in $\phi \wedge (\phi_1 \vee \phi_2) \wedge \neg\theta_i$. And, for such states, the fourth premise $\Xi \triangleright \Gamma; \Delta \Vdash \{\phi \wedge (\phi_1 \vee \phi_2) \wedge \neg\theta_i\}\ c_i\ \downarrow\{y : \tau \mid \psi\}$ ensures that the evaluation of c_i is well-defined, terminates, and the results in ψ.

•*Partial correctness rule for loop*

$$\frac{\begin{array}{l}\Xi \triangleright \Gamma, \Delta \vdash \{\phi_{\text{inv}}\}\ e\ \{y : \mathsf{B} \mid \mathsf{True}\} \quad \Xi \triangleright \Gamma, \Delta \vdash \{\phi_{\text{false}}\}\ e\ \{y : \mathsf{B} \mid y = \texttt{false}\} \\ \Xi \triangleright \Gamma, \Delta \vdash \{\phi_{\text{true}}\}\ e\ \{y : \mathsf{B} \mid y = \texttt{true}\} \quad \Xi \triangleright \Gamma; \Delta \Vdash \{\neg\phi_{\text{false}} \wedge \phi_{\text{inv}}\}\ c\ \{\phi_{\text{inv}}\}\end{array}}{\Xi \triangleright \Gamma; \Delta \Vdash \{\phi_{\text{inv}}\}\ \texttt{while}\ e\ \texttt{do}\ c\ \texttt{end}\ \{\phi_{\text{inv}} \wedge \neg\phi_{\text{true}}\}}$$

The formula ϕ_{inv} is an loop invariant which represents a property that is preserved throughout the iterations. The first premise $\Xi \triangleright \Gamma, \Delta \vdash \{\phi_{\text{inv}}\}\ e\ \{y : \mathsf{B} \mid \mathsf{True}\}$ ensures that any state in ϕ_{inv} makes the loop condition e be well-defined.

The second premise $\Xi \triangleright \Gamma, \Delta \vdash \{\phi_{\text{false}}\}\ e\ \{y : \mathsf{B} \mid y = \texttt{false}\}$ states that any state that yields a nondeterministic branch that results `true` in the evaluation of the loop condition e is in $\neg\phi_{\text{false}}$. Similarly, the third premise $\Xi \triangleright \Gamma, \Delta \vdash \{\phi_{\text{true}}\}\ e\ \{y : \mathsf{B} \mid y = \texttt{true}\}$ says that any state yields a nondeterministic branch that results `false` in the evaluation of the loop condition e is in $\neg\phi_{\text{true}}$.

The last premise $\Xi \triangleright \Gamma; \Delta \Vdash \{\neg\phi_{\text{false}} \wedge \phi_{\text{inv}}\}\ c\ \{\phi_{\text{inv}}\}$ says that for any state that satisfies the loop invariant and may result `true` in the evaluation of the loop condition, the execution of the loop body is well-defined and results only states which satisfy the loop invariant. Hence, this premise ensures that the loop invariant is really a loop invariant.

The consequence $\Xi \triangleright \Gamma; \Delta \Vdash \{\phi_{\text{inv}}\}\ \texttt{while}\ e\ \texttt{do}\ c\ \texttt{end}\ \{\phi_{\text{inv}} \wedge \neg\phi_{\text{true}}\}$ says when the loop is entered with an state satisfying the loop invariant, when the loop is escaped, the resulting states still satisfy the loop invariant. Since the loop can be escaped only if there is a nondeterministic branch that yields `false` in the evaluation of e, it is obvious that the states are also in $\neg\phi_{\text{true}}$

• *Total correctness rule for loop*

$$\frac{\begin{array}{l}\Xi \triangleright \Gamma, \Delta \vdash \{\phi_{\text{inv}}\}\ e\ \downarrow\{y : \mathsf{B} \mid \mathsf{True}\} \quad \Xi \triangleright \Gamma, \Delta \vdash \{\phi_{\text{false}}\}\ e\ \{y : \mathsf{B} \mid y = \texttt{false}\} \\ \Xi \triangleright \Gamma, \Delta \vdash \{\phi_{\text{true}}\}\ e\ \{y : \mathsf{B} \mid y = \texttt{true}\} \quad \Xi \triangleright \Gamma, \Delta \vdash \{\phi_{\text{exit}}\}\ e\ \downarrow\{y : \mathsf{B} \mid y = \texttt{false}\} \\ \Xi, z_0 : \mathsf{Z} \triangleright \Gamma; \Delta \Vdash \{\neg\phi_{\text{false}} \wedge \phi_{\text{inv}} \wedge \psi[z_0/z]\}\ c\ \downarrow\{\phi_{\text{inv}} \wedge \forall z : \mathsf{Z}\,.\,\psi \Rightarrow z < z_0\}\end{array}}{\Xi \triangleright \Gamma; \Delta \Vdash \{\phi_{\text{inv}}\}\ \texttt{while}\ e\ \texttt{do}\ c\ \texttt{end}\ \downarrow\{\phi_{\text{inv}} \wedge \neg\phi_{\text{true}}\}}$$

The side-condition: ψ is a well-formed formula under $\Xi, \Gamma, \Delta, z : \mathsf{Z}$ such that

$$\forall z : \mathsf{Z}\,.\,(\phi_{\text{inv}} \wedge \psi \wedge z < 0 \Rightarrow \phi_{\text{exit}}) \text{ and } \phi_{\text{inv}} \Rightarrow \exists! z : \mathsf{Z}\,.\,\psi$$

holds.

In addition to the partial correctness rule for while loops, now the first premise states that the loop invariant to guarantee the evaluation of the loop condition to terminate.

The fourth premise $\Xi \triangleright \Gamma, \Delta \vdash \{\phi_{\text{exit}}\}\ e\ \downarrow\{y : \mathsf{B} \mid y = \texttt{false}\}$ specifies ϕ_{exit} as a set of states that make the evaluation of the loop condition only be `false`. When a state is in ϕ_{exit}, it is promised that the loop terminates.

The side-condition ensures that the formula ψ represents a loop variant in the sense that for any state, there is a unique z that validates ψ. And, the side-condition also guarantees that when such z is negative, ψ implies ψ_{exit}. In other words, any state that the unique z is negative exits the loop.

The last premise $\Xi \triangleright \Gamma; \Delta \Vdash \{\neg\phi_{\text{false}} \wedge \phi_{\text{inv}} \wedge \psi[z_0]\}\ c\ \downarrow\{\phi_{\text{inv}}, \forall z.\ \psi \to z < z_0\}$, in addition to the partial correctness rule, ensures that the unique z decreases throughout iterations. Hence, at some point, ψ_{exit} gets validated.

The soundness of the proof rules

Theorem 5.1. The proof rules are sound. In other words. If a specification in any of the four forms is derived, its semantics holds.

Proof. Believing that the explanation in the presentation of each rule is convincing enough, let us formally prove the soundness of only the rules for loop and limit.

• *The soundness of the partial correctness rule for loop*

$$\frac{\begin{array}{l}\Xi \triangleright \Gamma, \Delta \vdash \{\phi_{\text{inv}}\}\ e\ \{y : \mathsf{B} \mid \mathsf{True}\} \quad \Xi \triangleright \Gamma, \Delta \vdash \{\phi_{\text{false}}\}\ e\ \{y : \mathsf{B} \mid y = \texttt{false}\} \\ \Xi \triangleright \Gamma, \Delta \vdash \{\phi_{\text{true}}\}\ e\ \{y : \mathsf{B} \mid y = \texttt{true}\} \quad \Xi \triangleright \Gamma; \Delta \Vdash \{\neg\phi_{\text{false}} \wedge \phi_{\text{inv}}\}\ c\ \{\phi_{\text{inv}}\}\end{array}}{\Xi \triangleright \Gamma; \Delta \Vdash \{\phi_{\text{inv}}\}\ \texttt{while}\ e\ \texttt{do}\ c\ \texttt{end}\ \{\phi_{\text{inv}} \wedge \neg\phi_{\text{true}}\}}$$

Consider any state (ξ, γ, δ) in $[\![\Xi, \Gamma, \Delta \Vdash \phi_{\text{inv}}]\!]$. The first premise ensures that $[\![\Gamma, \Delta \vdash e : \mathsf{B}]\!](\gamma, \delta)$ is not empty. If $tt \in [\![\Gamma, \Delta \vdash e : \mathsf{B}]\!](\gamma, \delta)$, by the second premise, $(\xi, \gamma, \delta) \in [\![\Xi, \Gamma, \Delta \Vdash \neg\phi_{\text{false}}]\!]$ holds. And, if $ff \in [\![\Gamma, \Delta \vdash e : \mathsf{B}]\!](\gamma, \delta)$, by the third premise, $(\xi, \gamma, \delta) \in [\![\Xi, \Gamma, \Delta \Vdash \neg\phi_{\text{true}}]\!]$ holds.

We first prove that $[\![\Gamma; \Delta \Vdash \texttt{while}\ e\ \texttt{do}\ c\ \texttt{end} : \mathsf{U}]\!]\gamma\,\delta$ is not $\mathfrak{e}$. By Remark 5.1-6, it holds when $[\![\Gamma; \Delta \Vdash A^i_{e,c} : \mathsf{U}]\!]\gamma\,\delta$ is not $\mathfrak{e}$ for any $i \in \mathbb{N}$.

Statement: for any $(\xi, \gamma, \delta) \in [\![\Xi, \Gamma, \Delta \Vdash \phi_{\text{inv}}]\!]$, it holds that $[\![\Gamma; \Delta \Vdash A^n_{e,c} : \mathsf{U}]\!]\gamma\,\delta \neq \mathfrak{e}$ for all $n \in \mathbb{N}$.

Proof. 1. (Base case) By definition, $[\![\Gamma; \Delta \Vdash A^0_{e,c} : \mathsf{U}]\!]\gamma\,\delta = \{\bot\} \neq \mathfrak{e}$ holds.

2. (Induction step)

$$[\![\Gamma; \Delta \Vdash A^{n+1}_{e,c} : \mathsf{U}]\!]\gamma\,\delta =$$
$$\bigcup_{b \in [\![\Gamma, \Delta \vdash e : \mathsf{B}]\!](\gamma,\delta)} \begin{cases} \bigcup_{r \in [\![\Gamma; \Delta \Vdash c : \mathsf{U}]\!]\gamma\,\delta} \begin{cases} [\![\Gamma; \Delta \Vdash A^n_{e,c} : \mathsf{U}]\!]\gamma\,\delta' & \text{if } r = (\delta', *), \\ \{\bot\} & \text{if } r = \bot, \\ \mathfrak{e} & \text{otherwise (if } r = \mathfrak{e}), \end{cases} & \text{if } b = tt, \\ (\delta, *) & \text{if } b = ff, \\ \{\bot\} & \text{if } b = \bot, \\ \mathfrak{e} & \text{otherwise (if } b = \mathfrak{e}). \end{cases}$$

is not $\mathfrak{e}$ if (i) $[\![\Gamma, \Delta \vdash e : \mathsf{B}]\!](\gamma, \delta)$ is not $\mathfrak{e}$, (ii) if $tt \in [\![\Gamma, \Delta \vdash e : \mathsf{B}]\!](\gamma, \delta)$, then $[\![\Gamma; \Delta \Vdash c : \mathsf{U}]\!]\gamma\,\delta$ is not $\mathfrak{e}$, and for all $(\delta', *) \in [\![\Gamma; \Delta \Vdash c : \mathsf{U}]\!]\gamma\,\delta$, $[\![\Gamma; \Delta \Vdash A^n_{e,c} : \mathsf{U}]\!](\gamma, \delta')$ is not $\mathfrak{e}$.

For the condition (i), since $(\xi, \gamma, \delta) \in [\![\Xi, \Gamma, \Delta \Vdash \phi_{\mathrm{inv}}]\!]$, the first premise ensures that $[\![\Gamma, \Delta \vdash e : \mathsf{B}]\!](\gamma, \delta)$ is not $\mathfrak{e}$. For the condition (ii), if $tt \in [\![\Gamma, \Delta \vdash e : \mathsf{B}]\!](\gamma, \delta)$, then by the second premise, $(\xi, \gamma, \delta) \in [\![\neg\phi_{\mathrm{false}}]\!]$ holds. Hence, by the last premise, $[\![\Gamma; \Delta \Vdash c : \mathsf{U}]\!]\gamma\,\delta$ is not $\mathfrak{e}$ and for all $(\delta', *) \in [\![\Gamma; \Delta \Vdash c : \mathsf{U}]\!]\gamma\,\delta$, it holds that $(\xi, \gamma, \delta') \in [\![\Xi, \Gamma, \Delta \vdash \phi_{\mathrm{inv}}]\!]$. Therefore, by the induction hypothesis, $[\![\Gamma; \Delta \Vdash A^n_{e,c} : \mathsf{U}]\!]\gamma'\,\delta$ is not $\mathfrak{e}$.

Therefore, $[\![\Gamma; \Delta \Vdash A^{n+1}_{e,c}\mathsf{U}]\!]\gamma\,\delta$ is not $\mathfrak{e}$.

□

Hence, by Remark 5.1-6 we only need to prove the following statement:

Statement: for any $(\xi, \gamma, \delta) \in [\![\Xi, \Gamma, \Delta \Vdash \phi_{\mathrm{inv}}]\!]$ and $(\delta', *) \in [\![\Gamma; \Delta \Vdash A^m_{e,c} : \mathsf{U}]\!]\gamma\,\delta$, it holds that $(\xi, \gamma, \delta') \in [\![\Xi, \Gamma, \Delta \Vdash \phi_{\mathrm{inv}} \wedge \neg\phi_{\mathrm{true}}]\!]$ for all $m \in \mathbb{N}$.

Proof. 1. (Base case) By the definition, $[\![\Gamma; \Delta \Vdash A^0_{e,c} : \mathsf{U}]\!]\gamma\,\delta = \{\bot\}$ holds.

2. (Induction step) We have seen that when $tt \in [\![\Gamma, \Delta \vdash e : \mathsf{B}]\!](\gamma, \delta)$, it holds that $(\xi, \gamma, \delta') \in [\![\Xi, \Gamma, \Delta \Vdash \phi_{\mathrm{inv}}]\!]$ where $(\delta', *) \in [\![\Gamma; \Delta \Vdash c : \mathsf{U}]\!]\gamma\,\delta$. Hence, by the induction hypothesis, we only need to show that if $ff \in [\![\Gamma, \Delta \vdash e : \mathsf{B}]\!](\gamma, \delta)$, then $(\xi, \gamma, \delta) \in [\![\Xi, \Gamma, \Delta \Vdash \phi_{\mathrm{inv}} \wedge \neg\phi_{\mathrm{true}}]\!]$ holds. It is trivial due to the observations made at the beginning of this proof.

□

• *The soundness of the total correctness rule for loop*

$$\frac{\begin{array}{l} \Xi \triangleright \Gamma, \Delta \vdash \{\phi_{\mathrm{inv}}\}\ e\ \downarrow\{y : \mathsf{B} \mid \mathsf{True}\} \quad \Xi \triangleright \Gamma, \Delta \vdash \{\phi_{\mathrm{false}}\}\ e\ \{y : \mathsf{B} \mid y = \mathtt{false}\} \\ \Xi \triangleright \Gamma, \Delta \vdash \{\phi_{\mathrm{true}}\}\ e\ \{y : \mathsf{B} \mid y = \mathtt{true}\} \quad \Xi \triangleright \Gamma, \Delta \vdash \{\phi_{\mathrm{exit}}\}\ e\ \downarrow\{y : \mathsf{B} \mid y = \mathtt{false}\} \\ \Xi, z_0 : \mathsf{Z} \triangleright \Gamma; \Delta \Vdash \{\neg\phi_{\mathrm{false}} \wedge \phi_{\mathrm{inv}} \wedge \psi[z_0/z]\}\ c\ \downarrow\{\phi_{\mathrm{inv}} \wedge \forall z : \mathsf{Z}\,.\,\psi \Rightarrow z < z_0\} \end{array}}{\Xi \triangleright \Gamma; \Delta \Vdash \{\phi_{\mathrm{inv}}\}\ \mathtt{while}\ e\ \mathtt{do}\ c\ \mathtt{end}\ \downarrow\{\phi_{\mathrm{inv}} \wedge \neg\phi_{\mathrm{true}}\}}$$

ψ is a well-formed formula under $\Xi, \Gamma, \Delta, z : \mathsf{Z}$ such that

$$\forall z : \mathsf{Z}\,.\,(\phi_{\mathrm{inv}} \wedge \psi \wedge z < 0 \Rightarrow \phi_{\mathrm{exit}}) \text{ and } \phi_{\mathrm{inv}} \Rightarrow \exists! z : \mathsf{Z}\,.\,\psi$$

holds.

Consider any state (ξ, γ, δ) in $[\![\Xi, \Gamma, \Delta \Vdash \phi_{\mathrm{inv}}]\!]$. By the proof of the soundness of the partial correctness rule, we only need to ensure that $\bot$ is not in $[\![\Gamma; \Delta \Vdash \texttt{while}\ e\ \texttt{do}\ \texttt{end} : \mathsf{U}]\!]\gamma\,\delta$. It holds if and only if there is $m \in \mathbb{N}$ such that $\bot \notin [\![\Gamma; \Delta \Vdash A^m_{e,c} : \mathsf{U}]\!]\gamma\,\delta$.

Let $c_{\xi,\gamma,\delta} \in \mathbb{Z}$ be the unique integer which satisfies $(\xi, \gamma, \delta, z \mapsto c_{\xi,\gamma,\delta}) \in [\![\Xi, \Gamma, \Delta, z{:}\mathsf{Z} \Vdash \psi]\!]$. The side-condition ensures that if $c_{\xi,\gamma,\delta} < 0$, then $(\xi, \gamma, \delta) \in [\![\Xi, \Gamma, \Delta \Vdash \psi_{\mathrm{exit}}]\!]$; hence, $[\![\Gamma, \Delta \vdash e : \mathsf{B}]\!](\gamma, \delta) = \{\mathit{ff}\}$ by the fourth premise. See that $c_{\xi,\gamma,\delta} < c_{\xi,\gamma,\delta'}$ if $(\xi, \gamma, \delta) \in [\![\Xi, \Gamma, \Delta \Vdash \phi_{\mathrm{inv}}]\!]$, $\mathit{tt} \in [\![\Gamma, \Delta \vdash e : \mathsf{B}]\!](\gamma, \delta)$, and $(\delta', *) \in [\![\Gamma; \Delta \Vdash c : \mathsf{U}]\!]\gamma\,\delta$. Hence, $\bot \notin [\![\Gamma; \Delta \Vdash A^{c_{\xi,\gamma,\delta}+1}_{e,c} : \mathsf{U}]\!]\gamma\,\delta$.

• *The soundness of the total correctness rule for limit*

$$\frac{\Xi, z : \mathsf{R} \triangleright \Gamma, x : \mathsf{Z} \vdash \{\phi'\}\ e\ \downarrow\{y : \mathsf{R} \mid \psi'\}}{\Xi \triangleright \Gamma \vdash \{\phi\}\ \texttt{lim}\, x\,.\,e\ \downarrow\{z : \mathsf{R} \mid \psi\}} \quad \begin{array}{l}\phi \Rightarrow \\ \exists z : \mathsf{R}\,.\,(\forall x : \mathsf{Z}\,.\,x \geq 0 \Rightarrow \phi' \wedge (\forall y : \mathsf{R}\,.\,\psi' \Rightarrow |y - z| \leq 2^{-x})) \wedge \psi\end{array}$$

Consider any $(\xi, \gamma) \in [\![\Xi, \Gamma \Vdash \phi]\!]$. Then, by the side-condition, we have $z_0 \in \mathbb{R}$ which makes $(\xi, z \mapsto z_0, \gamma)$ be in $[\![\Xi, z{:}\mathsf{R}, \Gamma \Vdash \forall x : \mathsf{Z}\,.\,\phi' \wedge (\forall y : \mathsf{R}\,.\,\psi' \Rightarrow |y - z| \leq 2^{-x})]\!]$ and in $[\![\Xi, z{:}\mathsf{R}, \Gamma \Vdash \psi]\!]$.

For any positive integer x_0, we have $(\xi, z \mapsto z_0, \gamma, x \mapsto x_0) \in [\![\Xi, z{:}\mathsf{R}, \Gamma, x{:}\mathsf{Z} \Vdash \phi']\!]$ and for any real number y_0, we have $(\xi, z \mapsto z_0, \gamma, \delta, x \mapsto x_0, y \mapsto y_0) \in [\![\Xi, z{:}\mathsf{R}, \Gamma, x{:}\mathsf{Z}, y{:}\mathsf{R} \Vdash \psi' \to |y - z| \leq 2^{-x}]\!]$. By the induction hypothesis, we have that $[\![\Gamma, x{:}\mathsf{Z} \vdash e : \mathsf{R}]\!](\gamma, x \mapsto x_0)$ (i) does not contain $\bot$, and (ii) each y' in $[\![\Gamma, x{:}\mathsf{Z} \vdash e : \mathsf{R}]\!](\gamma, x \mapsto x_0)$ satisfies $(\xi, z \mapsto z_0, \gamma, x \mapsto x_0, y \mapsto y') \in [\![\Xi, z{:}\mathsf{R}, \Gamma, x{:}\mathsf{Z}, y{:}\mathsf{R} \Vdash \psi']\!]$; hence, $(\xi, z \mapsto z_0, \gamma, x \mapsto x_0, y \mapsto y') \in [\![\Xi, z{:}\mathsf{R}, \Gamma, x{:}\mathsf{Z}, y{:}\mathsf{R} \Vdash |y - z| \leq 2^{-x}]\!]$. Therefore, $|y' - z_0| \leq 2^{-x_0}$.

To summarize, if $(\xi, \gamma) \in [\![\Xi, \Gamma \Vdash \phi]\!]$ and the side-condition hold, there is a real number z_0 where for any positive integer x_0, it holds that $[\![\Gamma, x{:}\mathsf{Z} \vdash e : \mathsf{R}]\!](\gamma, x \mapsto x_0)$ (i) does not contain $\bot$, and (ii) each y' in $[\![\Gamma, x{:}\mathsf{Z} \vdash e : \mathsf{R}]\!](\gamma, x \mapsto x_0)$ satisfies $|y' - z_0| \leq 2^{-x_0}$. Therefore, $[\![\Gamma \vdash \texttt{lim}\, x\,.\,e : \mathsf{R}]\!]\gamma = \{z_0\}$ and $(\xi, z \mapsto z_0, \gamma) \in [\![\Xi, z{:}\mathsf{R}, \Gamma \Vdash \psi]\!]$.

□

5.5 Example Formal Verifications

5.5.1 Abbreviations of Derivations

We soon prove the correctness of a specification using the proof rules. In order to make the proof a bit more readable, let us define the following abbreviations:

- $\{A\}\ e_1\ ?\{B\}\ e_2\ ?\{y : \tau \mid C\}$ denotes the specification $\{A\}\ e_1; e_2\ ?\{y : \tau \mid C\}$ that is derived from $\{A\}\ e_1\ ?\{B\}$ and $\{B\}\ e_2\ ?\{y : \tau \mid C\}$.
- $\{A\}\ \texttt{var}\ x := e\ \texttt{in}\ ?\{x : \tau \mid B\}\ c\ ?\{y : \tau \mid C\}$ where $x \notin FV(C)$ denotes the specification $\{A\}\ \texttt{var}\ x := e\ \texttt{in}\ c\ ?\{y : \tau \mid C\}$ derived from $\{A\}\ e\ ?\{x : \tau \mid B\}$ and $\{B\}\ c\ ?\{y : \tau \mid C\}$.
- $\{A\}\ e\ ?\{y : \tau \mid B\}\ ?\{y : \tau \mid C\}$ denotes the specification $\{A\}\ e\ ?\{y : \tau \mid C\}$ derived from $\{A\}\ e\ ?\{y : \tau \mid B\}$ and $B \Rightarrow C$. Similarly, $\{A\}\ \{B\}\ e\ ?\{y : \tau \mid C\}$ denotes the specification $\{A\}\ e\ ?\{y : \tau \mid C\}$ derived from $\{B\}\ e\ ?\{y : \tau \mid C\}$ and $A \Rightarrow B$.
- $\{A\}\ e\ \downarrow\{B\}\ \{B\}$ denotes the specification $\{A\}\ e\ \{B\}$ derived from $\{A\}\ e\ \downarrow\{B\}$.

5.5.2 Simple Arithmetical Expressions

Lemma 5.8. To each well-typed read-only expression $\Gamma \vdash e : \tau$ which is composed only of (1) constants, (2) coercions, (3) arithmetical operations, including $^{-1}$, (4) order comparisons, (5) guarded nondeterminisms, (6) conditionals, and (7) variables, there are $\Gamma \Vdash \phi$, $\Gamma \Vdash \phi^{\downarrow}$, $\Gamma \Vdash \psi_i$, and $\Gamma \vdash a_i : \tau$ for $i = 1 \cdots n$ for some n such that $\Gamma \vdash \phi^{\downarrow} \Rightarrow \phi$ holds and

$$\cdot \triangleright \Gamma \vdash \left\{\phi^{?}\right\}\ e\ ?\left\{y : \tau \mid (\psi_1 \wedge y = a_i) \vee \cdots \vee (\psi_n \wedge y = a_n)\right\}$$

is derivable. It can be recursively defined:

- $\cdot \triangleright \Gamma \vdash \left\{\mathsf{True}\right\}\ \texttt{skip}\ ?\left\{y : \mathsf{U} \mid y = \texttt{skip}\right\}$
- $\cdot \triangleright \Gamma \vdash \left\{\mathsf{True}\right\}\ \texttt{true}\ ?\left\{y : \mathsf{B} \mid y = \texttt{true}\right\}$
- $\cdot \triangleright \Gamma \vdash \left\{\mathsf{True}\right\}\ \texttt{false}\ ?\left\{y : \mathsf{B} \mid y = \texttt{false}\right\}$
- $\cdot \triangleright \Gamma \vdash \left\{\mathsf{True}\right\}\ k\ ?\left\{y : \mathsf{Z} \mid y = k\right\}$

  Suppose
  $$\cdot \triangleright \Gamma \vdash \left\{\phi_1^{?}\right\}\ e_1\ ?\left\{y : \tau \mid (\psi_1 \wedge y = a_1) \vee \cdots \vee (\psi_n \wedge y = a_n)\right\}$$
  $$\cdot \triangleright \Gamma \vdash \left\{\phi_2^{?}\right\}\ e_2\ ?\left\{y : \tau \mid (\theta_1 \wedge y = b_1) \vee \cdots \vee (\theta_m \wedge y = b_m)\right\}$$
- $\cdot \triangleright \Gamma \vdash \left\{\phi_1^{?}\right\}\ \iota(e_1)\ ?\left\{y : \mathsf{R} \mid (\psi_1 \wedge y = \iota(a_1)) \vee \cdots \vee (\psi_n \wedge y = \iota(a_n))\right\}$
- $\cdot \triangleright \Gamma \vdash \left\{\phi_1^{?} \wedge \phi_2^{?}\right\}\ e_1 \odot e_2\ ?\left\{y : \mathsf{Z} \mid \vee_{i,j}(\psi_i \wedge \theta_j \wedge y = a_i \odot b_j)\right\}$
- $\cdot \triangleright \Gamma \vdash \left\{\phi_1^{?} \wedge \phi_2^{?}\right\}\ e_1 \boxdot e_2\ ?\left\{y : \mathsf{R} \mid \vee_{i,j}(\psi_i \wedge \theta_j \wedge y = a_i \boxdot b_j)\right\}$
- $\cdot \triangleright \Gamma \vdash \left\{\phi\right\}\ {e_1}^{-1}\ \left\{y : \mathsf{R} \mid (\psi_1 \wedge y = {a_1}^{-1}) \vee \cdots \vee (\psi_n \wedge y = {a_n}^{-1})\right\}$
- $\cdot \triangleright \Gamma \vdash \left\{\phi^{\downarrow} \wedge a_1 \neq 0 \wedge \cdots \wedge a_n \neq 0\right\}\ e^{-1}\ \downarrow\left\{y : \mathsf{R} \mid (\psi_1 \wedge y = {a_1}^{-1}) \vee \cdots \vee (\psi_n \wedge y = {a_n}^{-1})\right\}$
- $\cdot \triangleright \Gamma \vdash \left\{\phi_1 \wedge \phi_2\right\}\ e_1 \hat{<} e_2\ \left\{y : \mathsf{B} \mid \vee_{i,j}(\psi_i \wedge \theta_j \wedge a_i < b_j \wedge y = \texttt{true}) \vee (\psi_i \wedge \theta_j \wedge a_i > b_j \wedge y = \texttt{false})\right\}$
- $$\cdot \triangleright \Gamma \vdash \begin{array}{l} \left\{\phi_1 \wedge \phi_2 \wedge (\wedge_{i,j} a_i \neq b_j)\right\} \\ \quad e_1 \hat{<} e_2 \\ \downarrow\left\{y : \mathsf{B} \mid \vee_{i,j}((\psi_i \wedge \theta_j \wedge a_i < b_j \wedge y = \texttt{true}) \vee (\psi_i \wedge \theta_j \wedge a_i > b_j \wedge y = \texttt{false}))\right\} \end{array}$$
- $\cdot \triangleright \Gamma \vdash \left\{\phi_1^{?} \wedge \phi_2^{?}\right\}\ e_1 < e_2\ ?\left\{y : \mathsf{B} \mid \vee_{i,j}((\psi_i \wedge \theta_j \wedge a_i < b_j \wedge y = \texttt{true}) \vee (\psi_i \wedge \theta_j \wedge a_i \geq b_j \wedge y = \texttt{false}))\right\}$
- $\cdot \triangleright \Gamma \vdash \left\{\phi_1^{?} \wedge \phi_2^{?}\right\}\ e_1 = e_2\ ?\left\{y : \mathsf{B} \mid \vee_{i,j}((\psi_i \wedge \theta_j \wedge a_i = b_j \wedge y = \texttt{true}) \vee (\psi_i \wedge \theta_j \wedge a_i \neq b_j \wedge y = \texttt{false}))\right\}$
- Suppose
  $$\cdot \triangleright \Gamma \vdash \left\{\phi_1^{?}\right\}\ e_1\ ?\left\{y : \mathsf{B} \mid (\psi_{\texttt{true}} \wedge y = \texttt{true}) \vee (\psi_{\texttt{false}} \wedge y = \texttt{false})\right\}$$
  $$\cdot \triangleright \Gamma \vdash \left\{\phi_2^{?}\right\}\ e_2\ ?\left\{y : \tau \mid (\psi_1 \wedge y = a_1) \vee \cdots \vee (\psi_n \wedge y = a_n)\right\}$$
  $$\cdot \triangleright \Gamma \vdash \left\{\phi_3^{?}\right\}\ e_3\ ?\left\{y : \tau \mid (\theta_1 \wedge y = b_1) \vee \cdots \vee (\theta_m \wedge y = b_m)\right\}$$

  $$\cdot \triangleright \Gamma \vdash \begin{array}{l} \left\{\phi_1^{?} \wedge (\psi_{\texttt{true}} \Rightarrow \phi_2^{?}) \wedge (\psi_{\texttt{false}} \Rightarrow \phi_3^{?})\right\} \\ \quad \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3\ \texttt{end} \\ ?\left\{y : \tau \mid (\vee_i(\psi_{\texttt{true}} \wedge \psi_i \wedge y = a_i)) \vee (\vee_j(\psi_{\texttt{false}} \wedge \theta_j \wedge y = b_j))\right\} \end{array}$$
- Suppose
  $$\cdot \triangleright \Gamma \vdash \left\{\phi_i^{?}\right\}\ e_i\ ?\left\{y : \mathsf{B} \mid (\psi^i_{\texttt{true}} \wedge y = \texttt{true}) \vee (\psi^i_{\texttt{false}} \wedge y = \texttt{false})\right\}$$
  $$\cdot \triangleright \Gamma \vdash \left\{\phi_i'^{?}\right\}\ e'_i\ ?\left\{y : \tau \mid (\psi^i_1 \wedge y = a^i_1) \vee \cdots \vee (\psi^i_{n_i} \wedge y = a^i_{n_i})\right\}$$

- $\cdot \triangleright \Gamma \vdash \big\{ \wedge_i (\phi_i \wedge (\psi^i_{\mathtt{true}} \Rightarrow \phi'_i)) \big\}\ \mathtt{case}\ e_1 \Rightarrow e'_1 \mid \cdots \mid e_n \Rightarrow e'_n\ \mathtt{end}\ \big\{ y : \tau \mid \vee_{i,j} (\psi^i_{\mathtt{true}} \wedge \psi^i_j \wedge y = a^i_j) \big\}$
  $\big\{ (\wedge_i (\phi_i \wedge (\psi^i_{\mathtt{true}} \Rightarrow \phi_i'^{\downarrow})) \wedge (\vee_i \phi_i^{\downarrow} \wedge \neg \psi^i_{\mathtt{false}})) \big\}$
- $\cdot \triangleright \Gamma \vdash \quad \mathtt{case}\ e_1 \Rightarrow e'_1 \mid \cdots \mid e_n \Rightarrow e'_n\ \mathtt{end}$
  $\downarrow \big\{ y : \tau \mid \vee_{i,j} (\psi^i_{\mathtt{true}} \wedge \psi^i_j \wedge y = a^i_j) \big\}$

Proof. We prove it constructively by induction on the well-typedness.

The cases of constants and variables are direct from the axioms.

• When the well-typedness is $\Gamma \vdash e_1 \odot e_2 : \mathsf{Z}$, by the induction hypothesis, we can derive the following specifications :

$$\cdot \triangleright \Gamma \vdash \big\{ \phi_1^? \big\}\ e_1\ ?\big\{ y : \mathsf{Z} \mid (\psi_1 \wedge y = a_1) \vee \cdots \vee (\psi_n \wedge y = a_n) \big\}$$
$$\cdot \triangleright \Gamma \vdash \big\{ \phi_2^? \big\}\ e_2\ ?\big\{ y : \mathsf{Z} \mid (\theta_1 \wedge y = b_1) \vee \cdots \vee (\theta_m \wedge y = b_m) \big\}$$

By applying the *rule for extending contexts*, the *rule of read-only variables*, and the *rule of postcondition weakening*, we can derive the specifications:

$$x_1{:}\mathsf{Z} \triangleright \Gamma \vdash \big\{ \phi_1 \wedge (\neg\psi_1 \vee x_1 \neq a_1) \wedge \cdots \wedge (\neg\psi_n \vee x_1 \neq a_n) \big\}\ e_1\ \big\{ y : \mathsf{Z} \mid y \neq x_1 \big\},$$
$$x_2{:}\mathsf{Z} \triangleright \Gamma \vdash \big\{ \phi_2 \wedge (\neg\theta_1 \vee x_2 \neq b_1) \wedge \cdots \wedge (\neg\theta_m \vee x_2 \neq b_m) \big\}\ e_2\ \big\{ y : \mathsf{Z} \mid y \neq x_2 \big\}.$$

Observe that $\phi_1^? \wedge \phi_2^? \wedge \neg\big(\phi_1 \wedge (\neg\psi_1 \vee x_1 \neq a_1) \wedge \cdots \wedge (\neg\psi_n \vee x_1 \neq a_n)\big) \wedge \neg\big(\phi_2 \wedge (\neg\theta_1 \vee x_2 \neq b_1) \wedge \cdots \wedge (\neg\theta_m \vee x_2 \neq b_m)\big) \Rightarrow \big((\psi_1 \wedge x_1 = a_1) \vee \cdots \vee (\psi_1 \wedge x_1 = a_n)\big) \wedge \big((\theta_1 \wedge x_2 = b_1) \vee \cdots \vee (\theta_1 \wedge x_2 = b_m)\big)$ holds. And, $\big((\theta_1 \wedge x_2 = b_1) \vee \cdots \vee (\theta_1 \wedge x_2 = b_m)\big) \Rightarrow \big(\vee_{i,j}\, \psi_i \wedge \theta_j \wedge y = a_i \odot b_j\big)[(x_1 \odot x_2)/y]$ holds.

By the *rule of postcondition weakening*, we can derive

$$\cdot \triangleright \Gamma \vdash \big\{ \phi_1^? \big\}\ e_1\ ?\big\{ y : \mathsf{Z} \mid \mathsf{True} \big\} \quad \text{and} \quad \cdot \triangleright \Gamma \vdash \big\{ \phi_2^? \big\}\ e_2\ ?\big\{ y : \mathsf{Z} \mid \mathsf{True} \big\}.$$

Therefore, using the *rule for integer arithmetic*, we get

$$\cdot \triangleright \Gamma \vdash \big\{ \phi_1^? \wedge \phi_2^? \big\}\ e_1 \odot e_2\ ?\big\{ y : \mathsf{Z} \mid \vee_{i,j} \psi_i \wedge \theta_j \wedge y = a_i \odot b_j \big\}.$$

• When the well-typedness is $\Gamma \vdash e_1 \boxdot e_2 : \mathsf{R}$, $\Gamma \vdash e_1 < e_2 : \mathsf{B}$, or $\Gamma \vdash e_1 = e_2 : \mathsf{B}$, it can be done identically.

• When the well-typedness is $\Gamma \vdash e^{-1} : \mathsf{R}$, we can derive the specification:

$$x{:}\mathsf{R} \triangleright \Gamma \vdash \big\{ \phi \wedge (\neg\psi_1 \vee x \neq a_1) \wedge \cdots \wedge (\neg\psi_n \vee x \neq a_n) \big\}\ e_1\ \big\{ y : \mathsf{R} \mid y \neq x \big\}$$

And, $\phi \wedge \neg\big(\phi \wedge (\neg\psi_1 \vee x \neq a_1) \wedge \cdots \wedge (\neg\psi_n \vee x \neq a_n)\big) \Rightarrow \big((\psi_1 \wedge y = {a_1}^{-1}) \vee \cdots \vee (\psi_n \wedge y = {a_n}^{-1})\big)[x^{-1}/y]$ holds. As $\cdot \triangleright \Gamma \vdash \big\{ \phi^? \big\}\ e\ ?\big\{ y : \mathsf{R} \mid \mathsf{True} \big\}$ is derivable by the *rule of postcondition weakening*, by the *rule for multiplicative inversions*, we have

$$\cdot \triangleright \Gamma \vdash \big\{ \phi \big\}\ e^{-1}\ \big\{ y : \mathsf{R} \mid (\psi_1 \wedge y = {a_1}^{-1}) \vee \cdots \vee (\psi_n \wedge y = {a_n}^{-1}) \big\}.$$

For the total correctness, see that $\phi^{\downarrow} \wedge a_1 \neq 0 \wedge \cdots \wedge a_n \neq 0 \wedge \neg\big(\phi \wedge (\neg\psi_1 \vee x \neq a_1) \wedge \cdots \wedge (\neg\psi_n \vee x \neq a_n)\big) \Rightarrow x \neq 0 \wedge \big((\psi_1 \wedge y = {a_1}^{-1}) \vee \cdots \vee (\psi_n \wedge y = {a_n}^{-1})\big)[x^{-1}/y]$ holds.

Therefore, using the *rule for multiplicative inversions*, we have

$$\cdot \triangleright \Gamma \vdash \big\{ \phi^{\downarrow} \wedge a_1 \neq 0 \wedge \cdots \wedge a_n \neq 0 \big\}\ e^{-1}\ \ \downarrow\big\{ y : \mathsf{R} \mid (\psi_1 \wedge y = {a_1}^{-1}) \vee \cdots \vee (\psi_n \wedge y = {a_n}^{-1}) \big\}.$$

See that $\phi^{\downarrow} \wedge a_1 \neq 0 \wedge \cdots \wedge a_n \neq 0 \Rightarrow \phi$ holds.

• When the well-typedness is $\Gamma \vdash e_1 \hat{<} e_2 : \mathsf{B}$, it can be done similarly. • When the well-typedness is $\Gamma \vdash \textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3 \texttt{ end} : \tau$, we can derive the specifications:

$$\cdot \triangleright \Gamma \vdash \{\phi_1^?\}\ e_1\ ?\{y : \mathsf{B} \mid (\psi_{\texttt{true}} \wedge y = \texttt{true}) \vee (\psi_{\texttt{false}} \wedge y = \texttt{false})\},$$
$$\cdot \triangleright \Gamma \vdash \{\phi_2^?\}\ e_2\ ?\{y : \tau \mid (\psi_1 \wedge y = a_1) \vee \cdots \vee (\psi_n \wedge y = a_n)\},$$
$$\cdot \triangleright \Gamma \vdash \{\phi_3^?\}\ e_3\ ?\{y : \tau \mid (\theta_1 \wedge y = b_1) \vee \cdots \vee (\theta_m \wedge y = b_m)\}.$$

By the *rule of read-only variables*, we can derive specifications:

$$\cdot \triangleright \Gamma \vdash \{\phi_1 \wedge \neg\psi_{\texttt{true}}\}\ e_1\ \{y : \mathsf{B} \mid y = \texttt{false}\}$$
$$\cdot \triangleright \Gamma \vdash \{\phi_1 \wedge \neg\psi_{\texttt{false}}\}\ e_1\ \{y : \mathsf{B} \mid y = \texttt{true}\}$$

Let $\psi \coloneqq (\psi_{\texttt{true}} \wedge \psi_1 \wedge y = a_1) \vee \cdots \vee (\psi_{\texttt{true}} \wedge \psi_n \wedge y = a_n) \vee (\psi_{\texttt{false}} \wedge \theta_1 \wedge y = b_1) \vee \cdots \vee (\psi_{\texttt{false}} \wedge \theta_m \wedge y = b_m)$.

See that by the *rule for read-only variables*, and the *rule for postcondition weakening*, we can derive

$$\cdot \triangleright \Gamma \vdash \{\phi_2^? \wedge \psi_{\texttt{true}}\}\ e_2\ ?\{y : \tau \mid \psi\} \quad \text{and} \quad \cdot \triangleright \Gamma \vdash \{\phi_3^? \wedge \psi_{\texttt{false}}\}\ e_3\ ?\{y : \tau \mid \psi\}.$$

By the *rule of precondition strengthening*, we can derive the specifications:

$$\cdot \triangleright \Gamma \vdash \{(\phi_1^? \wedge (\psi_{\texttt{true}} \Rightarrow \phi_2^?) \wedge (\psi_{\texttt{false}} \Rightarrow \phi_3^?)) \wedge \neg(\phi_1 \wedge \neg\psi_{\texttt{true}})\}\ e_2\ ?\{y : \tau \mid \psi\}$$
$$\cdot \triangleright \Gamma \vdash \{(\phi_1^? \wedge (\psi_{\texttt{true}} \Rightarrow \phi_2^?) \wedge (\psi_{\texttt{false}} \Rightarrow \phi_3^?)) \wedge \neg(\phi_1 \wedge \neg\psi_{\texttt{false}})\}\ e_3\ ?\{y : \tau \mid \psi\},$$

Therefore, using the *rule for conditional*, we get

$$\cdot \triangleright \Gamma \vdash \{\phi_1^? \wedge (\psi_{\texttt{true}} \Rightarrow \phi_2^?) \wedge (\psi_{\texttt{false}} \Rightarrow \phi_3^?)\}\ \texttt{if } e_1 \texttt{ then } e_2 \texttt{ else } e_3 \texttt{ end}\ ?\{y : \tau \mid \psi\}$$

Check that $\phi_1^{\downarrow} \wedge (\psi_{\texttt{true}} \Rightarrow \phi_2^{\downarrow}) \wedge (\psi_{\texttt{false}} \Rightarrow \phi_3^{\downarrow}) \Rightarrow \phi_1 \wedge (\psi_{\texttt{true}} \Rightarrow \phi_2) \wedge (\psi_{\texttt{false}} \Rightarrow \phi_3)$ holds.

• When the well-typedness is $\Gamma \vdash \texttt{case } e_1 \Rightarrow e_1' \mid \cdots \mid e_d \Rightarrow e_d' \texttt{ end} : \tau$,

$$\cdot \triangleright \Gamma \vdash \{\phi_i^?\}\ e_i\ ?\{y : \mathsf{B} \mid (\psi^i_{\texttt{true}} \wedge y = \texttt{true}) \vee (\psi^i_{\texttt{false}} \wedge y = \texttt{false})\},$$
$$\cdot \triangleright \Gamma \vdash \{\phi_i'^?\}\ e_i'\ ?\{y : \tau \mid (\psi_1^i \wedge y = a_1^i) \vee \cdots \vee (\psi_{n_i}^i \wedge y = a_{n_i}^i)\}$$

Therefore, for all i, by the *rule for read-only variables*, we have

$$\cdot \triangleright \Gamma \vdash \{\phi_i^? \wedge \neg\psi^i_{\texttt{true}}\}\ e_i\ ?\{y : \mathsf{B} \mid \psi^i_{\texttt{false}} \wedge y = \texttt{false}\},$$

Let us define $\psi \coloneqq \vee_{i,j}(\psi^i_{\texttt{true}} \wedge \psi^i_j \wedge y = a^i_j)$. By the *rule for read-only variables* and the *rule for precondition strengthening*, we have

$$\cdot \triangleright \Gamma \vdash \{\phi_i'^? \wedge \psi^i_{\texttt{true}}\}\ e_i'\ ?\{y : \tau \mid \psi\}$$

Define $\phi^? \coloneqq \wedge_i \phi_i \wedge (\psi^i_{\texttt{true}} \Rightarrow \phi_i'^?)$. See that $\phi^? \wedge \neg(\phi_i \wedge \neg\psi^i_{\texttt{true}}) \Rightarrow \phi_i'^? \wedge \psi^i_{\texttt{true}}$ holds. Hence, using the *rule of precondition strengthening*, we get

$$\cdot \triangleright \Gamma \vdash \{\phi^? \wedge \neg(\phi_i \wedge \neg\psi^i_{\texttt{true}})\}\ e_i'\ ?\{y : \tau \mid \psi\}$$

Finally, using the *rule for guarded cases*, we get

$$\cdot \triangleright \Gamma \vdash \{\wedge_i \phi_i \wedge (\psi^i_{\texttt{true}} \Rightarrow \phi_i')\}\ \texttt{case } e_1 \Rightarrow e_1' \mid \cdots \mid e_n \Rightarrow e_n' \texttt{ end}\ \{y : \tau \mid \vee_{i,j}(\psi^i_{\texttt{true}} \wedge \psi^i_j \wedge y = a^i_j)\}$$

For the total correctness, see that we have the specification:

$$\cdot \triangleright \Gamma \vdash \{\phi_i^{\downarrow} \wedge \neg\psi^i_{\texttt{false}}\}\ e_i\ \ \downarrow\{y : \mathsf{B} \mid y = \texttt{true}\}$$

By applying the *rule for precondition strengthening*, we can derive the specification:

$$\cdot \triangleright \Gamma \vdash \left\{\phi^{\downarrow} \wedge (\vee_i \phi_i^{\downarrow} \wedge \neg \psi^i_{\texttt{false}}) \wedge \neg(\phi_i \wedge \neg \psi^i_{\texttt{true}})\right\}\ e'_i\ \ \downarrow\left\{y : \tau \mid \psi\right\}$$

Therefore, by the *rule for guarded cases*, we have

$$\begin{array}{l}\cdot \triangleright \Gamma \vdash \\ \qquad \left\{ \wedge_i \phi_i \wedge (\psi^i_{\texttt{true}} \Rightarrow \phi_i'^{\downarrow}) \wedge (\vee_i \phi_i^{\downarrow} \wedge \neg \psi^i_{\texttt{false}})\right\} \\ \qquad\quad \texttt{case}\ e_1 \Rightarrow e'_1 \mid \cdots \mid e_n \Rightarrow e'_n\ \texttt{end} \\ \qquad\quad \downarrow\left\{y : \tau \mid \vee_{i,j} (\psi^i_{\texttt{true}} \wedge \psi^i_j \wedge y = a^i_j)\right\}\end{array}$$

□

When we further restrict expressions that are constructed only by (1) constants, (2) coercion, (3) arithmetical operations including $^{-1}$, and (4) variables, i.e., when $\Gamma \vdash e : \tau$ is also a term in the assertion language, the following specifications are derivable:

$$\cdot \triangleright \Gamma \vdash \left\{\mathsf{True}\right\}\ e\ \left\{y : \tau \mid y = e\right\}$$

and

$$\cdot \triangleright \Gamma \vdash \left\{a_1 \neq 0 \wedge \cdots \wedge a_n \neq 0\right\}\ e\ \ \downarrow\left\{y : \tau \mid y = e\right\}$$

where a_i are the subexpressions of e that appears in the form ${a_i}^{-1}$.

And, when $\Gamma \vdash e : \tau$ is specified as

$$\cdot \triangleright \Gamma, \Delta, x{:}\tau \vdash \left\{\phi^{?}\right\}\ e\ \ ?\left\{y : \tau \mid (\psi_1 \wedge y = a_1) \vee \cdots \vee (\psi_n \wedge y = a_n)\right\},$$

where y does not appear free in ψ_i, the specification

$$\cdot \triangleright \Gamma; \Delta, x : \tau \Vdash \left\{\phi^{?} \wedge (\psi_1 \Rightarrow \psi[a_1/x]) \wedge \cdots \wedge (\psi_n \Rightarrow \psi[a_n/x])\right\}\ x \coloneqq e\ \ ?\left\{\psi\right\}$$

is derivable.

5.5.3 Formal Verification of Computing π

The expression in Figure 5.4 which we abbreviate as pi computes π by searching for the root of the sine function. In this section, we prove the correctness of the expression using our verification principles. For this section, assume that our assertion language is strong enough to do basic mathematical analysis. And, in order to simplify our presentation, let us hide explicit type distinctions in the assertion language that we write $\Xi \triangleright \Gamma \vdash \left\{\mathsf{True}\right\}\ y \times \iota(4+3)\ \downarrow\left\{z : \mathsf{R} \mid z = y \times (4+3)\right\}$ to refer to $\Xi \triangleright \Gamma \vdash \left\{\mathsf{True}\right\}\ y \times \iota(4+3)\ \downarrow \left\{z : \mathsf{R} \mid z = y \times \iota(4+3)\right\}$.

We also assume that the abbreviations $\mathsf{abs}(x)$ and $\mathsf{prec}(x)$ are correct without proving.

In order to prove the correctness of the program by parts, let us abbreviate the subexpressions as follows:

- w_{in}: the inner while loop at Line 10-21
- e_{in}: the loop condition of w_{in} at Line 10-15
- sine: the inner limit expression at Line 5-21
- sine_approx: the expression of sine such that sine is `lim` p . sine_approx

```
1   lim(q,  var δ := prec(−q) in
2           var a := ι(3) in var b := ι(4) in
3           while case b − a ?> δ/ι(2) ⇒ true | b − a ?< δ ⇒ false end do
4             var x := (b − a)/ι(2) in
5             if (lim p. var ϵ := prec(−p) in
6                       var n := 0 in
7                       var s := ι(1) in
8                       var r := ι(0) in
9                       var e := x in
10                      while (if e ?> ι(0)
11                              then
12                                case ι(2) × e ?> ϵ ⇒ true | e ?< ϵ ⇒ false end
13                              else
14                                case − ι(2) × e ?> ϵ ⇒ true | −e ?< ϵ ⇒ false end
15                              end)
16                       do
17                         n := n + 1;
18                         r := r + e × s;
19                         s := s × ι(−1);
20                         e := e × x × x/ι(2 × n + 1)/ι(2 × n)
21                       end; r) ?> ι(0) then a := x else b := x
22              end
23          end; a)
```

Figure 5.4: A Clerical expression for computing π.

- w_{out}: the outer while loop at Line 3 - 23
- e_{out}: the loop condition of w_{out}
- bisect: the if-then-else at Line 5-22
- pi the entire limit expression
- pi_approx the expression of pi such that pi is `lim` q . pi_approx

We start the proof by verifying the correctness of sine. The sine function is computed using the fact that the series can approximate the (mathematical) value $\sin(x)$:

$$|\sin(x) - \Sigma_{n=0}^{m} s(n) \cdot q(n,x)| \leq |q(m+1,x)| \quad \text{where} \quad q(n,x) := \frac{(-1)^n x^{2n+1}}{(2n+1)!} \text{ and } s(n) = (-1)^n \ .$$

Let us denote the partial sum $A(m,x) := \Sigma_{n=0}^{m} s(n) \cdot q(n,x)$ and let it be defined at $A(-1,x) := 0$.

First, we want to show that sine_approx computes 2^{-p} approximation to the $\sin(x)$. In order to prove it, we need to reason on the while loop w_{in}.

In order to reason on the condition of the loop, let us define the three formulae

$\phi_{\text{true}} \coloneqq \epsilon > 0 \wedge \lvert e \rvert \geq \epsilon$	e_{in} evaluates only to `true`
$\phi_{\text{false}} \coloneqq \epsilon > 0 \wedge 2 \times \lvert e \rvert \leq \epsilon$	e_{in} evaluates only to `false`
$\phi_{\text{exit}} \coloneqq \phi_{\text{false}}$	e_{in} terminates and evaluates only to `false`

Applying Lemma 5.8, we can easily verify

$$\begin{array}{lll} \{\phi_{\text{true}}\} & \mathsf{e}_{\text{in}} & \{y : \mathsf{B} \mid y = \texttt{true}\} \\ \{\phi_{\text{false}}\} & \mathsf{e}_{\text{in}} & \{y : \mathsf{B} \mid y = \texttt{false}\} \\ \{\phi_{\text{exit}}\} & \mathsf{e}_{\text{in}} & \downarrow\{y : \mathsf{B} \mid y = \texttt{false}\} \end{array}$$

Let us define

$$\phi_{\text{inv}} \coloneqq n \geq 0 \wedge \epsilon = 2^{-p} \wedge e = q(n,x) \wedge s = s(n) \wedge r = A(n-1,x) \wedge \epsilon = 2^{-p}$$

as a candidate for a loop-invariant. As the assigned expressions are simple arithmetical, we can keep substituting the expressions backwards to get the specifications:

$$\begin{array}{l} \{n+1 \geq 0 \wedge \epsilon = 2^{-p} \wedge e \times x^2/\big((2\times n+3)\times(2\times n+2)\big) = q(n+1,x) \\ \qquad \wedge -s = s(n+1) \wedge r + e \times s = A(n,x) \wedge 2\times n+3 \neq 0 \wedge 2\times n+2 \neq 0\} \\ \quad n := n+1 \\ \downarrow\{n \geq 0 \wedge \epsilon = 2^{-p} \wedge e \times x^2/\big((2\times n+1)\times(2\times n)\big) = q(n,x) \wedge -s = s(n) \wedge r + e \times s = A(n-1,x) \\ \qquad \wedge 2\times n+1 \neq 0 \wedge 2\times n \neq 0\} \\ \quad r := r + e \times s \\ \downarrow!\{n \geq 0 \wedge \epsilon = 2^{-p} \wedge e \times x^2/\big((2\times n+1)\times(2\times n)\big) = q(n,x) \wedge -s = s(n) \wedge r = A(n-1,x) \\ \qquad \wedge 2\times n+1 \neq 0 \wedge 2\times n \neq 0\} \\ \quad s := s \times \iota(-1) \\ \downarrow\{n \geq 0 \wedge \epsilon = 2^{-p} \wedge e \times x^2/\big((2\times n+1)\times(2\times n)\big) = q(n,x) \wedge s = s(n) \wedge r = A(n-1,x) \\ \qquad \wedge 2\times n+1 \neq 0 \wedge 2\times n \neq 0\} \\ \quad e := e \times x \times x/\iota(2\times n+1)/\iota(2\times n) \\ \downarrow\{\phi_{\text{inv}}\} \end{array}$$

By checking the implication $\phi_{\text{inv}} \Rightarrow \big(n+1 \geq 0 \wedge \epsilon = 2^{-p} \wedge e \times x^2/\big((2\times n+3)\times(2\times n+2)\big) = q(n+1,x) \wedge -s = s(n+1) \wedge r + e \times s = A(n,x) \wedge \wedge 2\times n+3 \neq 0 \wedge 2\times n+2 \neq 0\big)$, we confirm that ϕ_{inv} is indeed a loop-invariant.

For a formula θ, let us write

$$\mathsf{min}(z,\theta) \coloneqq \theta \wedge \forall z'.\ \theta \Rightarrow z \leq z'.$$

Namely, z satisfies $\mathsf{min}(z,\theta)$ if and only if z is the smallest number that satisfies θ. Let us define

$$\psi \coloneqq \mathsf{min}(z, \forall m.\ m \geq z+n \Rightarrow |q(m,x)| \leq \epsilon/2)$$

for our loop-invariant. I.e., z satisfies ψ if and only if z is the smallest distance to the index m where all terms beyond the index $|q(m',x)|, m' \geq m$ satisfy the exit condition $|q(m',x)| \leq \epsilon/2$. For any n, x, the fact on the power series ensures the existence of such z and being "the smallest" guarantees the uniqueness. For the other side-condition, when $\phi_{\text{inv}} \wedge \psi \wedge z < 0$ holds, by ψ, $|q(n,x)| \leq \epsilon/2$ holds. And, by ϕ_{inv}, $e = q(n,x)$ holds. Hence, we have $\epsilon > 0 \wedge |e| \leq \epsilon/2$ which is ψ_{exit}.

The only programming variable that ψ refers to and gets modified in the loop is n. Hence, we only need to check the implication:

$$\mathsf{min}(z_0, \forall m.\ m \geq z_0+n \Rightarrow |q(m,x)| \leq \epsilon/2) \Rightarrow \big(\mathsf{min}(z, \forall m.\ m \geq z+n+1 \Rightarrow |q(m,x)| \leq \epsilon/2) \Rightarrow z < z_0\big)\ .$$

Let m_0 be the smallest index such that $\forall m > m_0.\ |q(m,x)| \leq \epsilon/2$ holds. Assuming z s.t. $\mathsf{min}(z, \forall m.\ m \geq z+n+1 \Rightarrow |q(m,x)| \leq \epsilon/2)$ and z_0 s.t. $\mathsf{min}(z_0, \forall m.\ m \geq z_0+n \Rightarrow |q(m,x)| \leq \epsilon/2)$, we get $m_0 = z+n+1 = z_0+n$. Hence, $z < z_0$ holds. With the implication, we derive the triple to confirm that ψ is a loop-variant:

$$\begin{array}{l}
\big\{\mathsf{min}(z_0, \forall m.\ m \geq z_0+n \Rightarrow |q(m,x)| \leq \epsilon/2)\big\} \\
\big\{\forall z.\ \mathsf{min}(z, \forall m.\ m \geq z+n+1 \Rightarrow |q(m,x)| \leq \epsilon/2) \Rightarrow z < z_0\big\} \\
\quad n := n+1 \\
\downarrow\big\{\forall z.\ \mathsf{min}(z, \forall m.\ m \geq z+n \Rightarrow |q(m,x)| \leq \epsilon/2) \Rightarrow z < z_0\big\}
\end{array}$$

Therefore, using the *rule for conjunctions of assertions*, we get all the premises to apply the rule for loops. Applying it with the *rule for sequential composition*, we derive the following specification:

$$\big\{\phi_{\text{inv}}\big\}\ \mathsf{w}_{\text{in}}; r\ \downarrow\big\{y : \mathsf{R} \mid |\sin(x) - y| < 2^{-p}\big\}\ .$$

Again, since the expressions assigned to the variables before w_{in} at Line 5-9 are simple arithmetical, by substituting backwards, we get the specifications derived.

$\{\mathsf{True}\}$
$\{0 \geq 0 \wedge 2^{-p} = 2^{-p} \wedge x = q(0,x) \wedge 1 = s(0) \wedge 0 = A(-1,x)\}$
  $\mathtt{var}\ \epsilon := \mathsf{prec}(p)\ \mathtt{in}$
$\downarrow\{\epsilon : \mathsf{R} \mid 0 \geq 0 \wedge \epsilon = 2^{-p} \wedge x = q(0,x) \wedge 1 = s(0) \wedge 0 = A(-1,x)\}$
  $\mathtt{var}\ n := 0\ \mathtt{in}$
$\downarrow\{n : \mathsf{Z} \mid n \geq 0 \wedge \epsilon = 2^{-p} \wedge x = q(n,x) \wedge 1 = s(n) \wedge 0 = A(n-1,x)\}$
  $\mathtt{var}\ s := \iota(1)\ \mathtt{in}$
$\downarrow\{s : \mathsf{R} \mid n \geq 0 \wedge \epsilon = 2^{-p} \wedge x = q(n,x) \wedge s = s(n) \wedge 0 = A(n-1,x)\}$
  $\mathtt{var}\ r := \iota(0)\ \mathtt{in}$
$\downarrow\{r : \mathsf{R} \mid n \geq 0 \wedge \epsilon = 2^{-p} \wedge x = q(n,x) \wedge s = s(n) \wedge r = A(n-1,x)\}$
  $\mathtt{var}\ e := x\ \mathtt{in}$
$\downarrow\{e : \mathsf{R} \mid n \geq 0 \wedge \epsilon = 2^{-p} \wedge e = q(n,x) \wedge s = s(n) \wedge r = A(n-1,x)\}$
  $\mathsf{w}_{\mathsf{in}}; r$
$\downarrow\{y : \mathsf{R} \mid |\sin(x) - y| < 2^{-p}\}$

In consequence, we can specify the expression in the limit expression at Line 5-21:

$$\Xi \triangleright p : \mathsf{Z}, x : \mathsf{R} \vdash \{\mathsf{True}\}\ \ \mathsf{sine\_approx}\ \ \downarrow\{y : \mathsf{R} \mid |\sin(x) - y| < 2^{-p}\}.$$

Now, we prove that the limit expression computes $\sin(x)$. Let $\psi := z = \sin(x)$ and $\psi' := |y - \sin(x)| \leq 2^{-p}$. Since $\exists z : \mathsf{R}\,.\,\forall p : \mathsf{Z}\,.\,p > 0 \Rightarrow \forall y : \mathsf{R}\,.\,\big(|\sin(x) - y| < 2^{-p} \Rightarrow |y - z| \leq 2^{-p}\big) \wedge z = \sin(x)$ holds by simply letting $z = \sin(x)$, using the *rule for limit*, we can derive the specification:

$$\Xi \triangleright x : \mathsf{R} \vdash \{\mathsf{True}\}\ \ \mathsf{sine}\ \ \downarrow\{z : \mathsf{R} \mid z = \sin(x)\}\ .$$

The correctness of the entire expression is based on the fact that $\sin(x)$ is strictly decreasing in $[3,4]$ admitting $\sin(x) = 0$ at $x = \pi$, $\sin(3) > 0 > \sin(4)$, and $\pi \notin \mathbb{Q}$.

In order to prove a specification of the loop at Line 3-23, let us define

$\phi_{\mathrm{true}} := a < b \wedge \delta > 0 \wedge b - a \geq \delta$	$\mathsf{e}_{\mathrm{out}}$ evaluates only to `true`
$\phi_{\mathrm{false}} := a < b \wedge \delta > 0 \wedge 2 \times (b-a) \leq \delta$	$\mathsf{e}_{\mathrm{out}}$ evaluates only to `false`
$\phi_{\mathrm{exit}} := \phi_{\mathrm{false}}$	$\mathsf{e}_{\mathrm{out}}$ terminates and evaluates only to `false`.

Again, by Lemma 5.8, we can easily derive

$$\begin{array}{lll} \{\phi_{\mathrm{true}}\} & \mathsf{e}_{\mathrm{out}} & \{y : \mathsf{B} \mid y = \mathtt{true}\}\ . \\ \{\phi_{\mathrm{false}}\} & \mathsf{e}_{\mathrm{out}} & \{y : \mathsf{B} \mid y = \mathtt{false}\}\ , \\ \{\phi_{\mathrm{exit}}\} & \mathsf{e}_{\mathrm{out}} & \downarrow\{y : \mathsf{B} \mid y = \mathtt{false}\}\ . \end{array}$$

Let

$$\phi_{\mathrm{inv}} := a, b \in \mathbb{Q} \wedge 3 \leq a < \pi < b \leq 4 \wedge \delta = 2^q$$

be the candidate for a loop-invariant and

$$\psi := \mathsf{min}(z, b - a \leq 2^{z-q-1})$$

be the candidate for a loop-variant. See that for any a, b, q such that $b - a > 0$, there exists unique z that satisfies ψ. And, when such z is negative, $2 \times (b-a) < 2^q$ holds, hence satisfies ϕ_{exit}.

Recall the specification:

$$\left\{x' = x\right\}\ \mathsf{sine}\ \left\{y : \mathsf{R} \mid y = \sin(x) \wedge x' = x\right\}$$

Since $x_1 \neq \sin(x')$ does not refer to any programming variables, we can add it in the precondition and postcondition and reduce it to the following specification:

$$\left\{x_1 \neq \sin(x)\right\}\ \mathsf{sine}\ \left\{y : \mathsf{R} \mid y \neq x_1\right\}$$

Similarly, we have $\left\{x_2 \neq 0\right\}\ 0\ \downarrow\left\{y : \mathsf{R} \mid y \neq x_2\right\}$. Since $\sin(x) \geq 0 \wedge x_1 = \sin(x) \wedge x_2 = 0 \Rightarrow (x_1 > x_2 \Rightarrow \mathtt{true} = \mathtt{true}) \wedge (x_2 > x_1 \Rightarrow \mathtt{false} = \mathtt{true})$ and $\sin(x) \leq 0 \wedge x_1 = \sin(x) \wedge x_2 = 0 \Rightarrow (x_1 > x_2 \Rightarrow \mathtt{false} = \mathtt{true}) \wedge (x_2 > x_1 \Rightarrow \mathtt{true} = \mathtt{true})$ hold, we can derive the specifications:

$$\left\{\sin(x) \geq 0\right\}\ \mathsf{sine}\ \hat{>}\ 0\ \left\{y : \mathsf{R} \mid y = \mathtt{true}\right\} \quad \text{and} \quad \left\{\sin(x) \leq 0\right\}\ \mathsf{sine}\ \hat{>}\ 0\ \left\{y : \mathsf{R} \mid y = \mathtt{false}\right\}.$$

Since $a, b \in \mathbb{Q} \wedge 3 \leq a < \pi < b \leq 4 \wedge \delta = 2^q \wedge x = (a+b)/2 \wedge x_1 = \sin(x) \wedge x_2 = 0 \Rightarrow x_1 \neq x_2$ by the fact that $\pi \notin \mathbb{Q}$ is the unique root of $\sin(x)$ in $[3, 4]$, we can derive

$$\left\{a, b \in \mathbb{Q} \wedge 3 \leq a < \pi < b \leq 4 \wedge \delta = 2^q \wedge x = (a+b)/2\right\}\ \mathsf{sine}\ \downarrow\left\{\mathsf{True}\right\}.$$

By Intermediate Value Theorem, when $\sin(x) > 0$, $x = (a+b)/2$, and $3 \leq a < \pi < b \leq 4$, we can refine $3 \leq x < \pi < b \leq 4$. And, under the condition,

$$\mathsf{min}(z_0, b - a \leq 2^{z_0 - q - 1}) \Rightarrow \forall z.\ (\mathsf{min}(z, b - x \leq 2^{z - q - 1})) \Rightarrow z < z_0$$

holds. Hence, using the implications, we derive the following specification:

$$\begin{array}{l}
\left\{a, b \in \mathbb{Q} \wedge 3 \leq a < \pi < b \leq 4 \wedge \delta = 2^q \wedge x = (a+b)/2 \wedge \sin(x) > 0 \wedge \mathsf{min}(z_0, b - a \leq 2^{z_0 - q - 1})\right\} \\
\left\{x, b \in \mathbb{Q} \wedge 3 \leq x < \pi < b \leq 4 \wedge \delta = 2^q \wedge \forall z.\ (\mathsf{min}(z, b - x \leq 2^{z - q - 1})) \Rightarrow z < z_0\right\} \\
\quad a := x \\
\downarrow\left\{a, b \in \mathbb{Q} \wedge 3 \leq a < \pi < b \leq 4 \wedge \delta = 2^q \wedge \forall z.\ (\mathsf{min}(z, b - a \leq 2^{z - q - 1})) \Rightarrow z < z_0\right\}
\end{array}$$

We can derive a similar specification for the other branch. And, using them for the *rule for conditional* yields the specification:

$$\begin{array}{l}
\left\{a, b \in \mathbb{Q} \wedge 3 \leq a < \pi < b \leq 4 \wedge \delta = 2^q \wedge x = (a+b)/2 \wedge \mathsf{min}(z_0, b - a \leq 2^{z_0 - q - 1})\right\} \\
\quad \mathsf{bisect} \\
\downarrow\left\{a, b \in \mathbb{Q} \wedge 3 \leq a < \pi < b \leq 4 \wedge \delta = 2^q \wedge \forall z.\ (\mathsf{min}(z, b - a \leq 2^{z - q - 1})) \Rightarrow z < z_0\right\}
\end{array}$$

Since the postcondition of the above specification does not refer to x, we can derive the following specification:

$$\begin{array}{l}
\left\{a, b \in \mathbb{Q} \wedge 3 \leq a < \pi < b \leq 4 \wedge \delta = 2^q \wedge \mathsf{min}(z_0, b - a \leq 2^{z_0 - q - 1})\right\} \\
\left\{a, b \in \mathbb{Q} \wedge 3 \leq a < \pi < b \leq 4 \wedge \delta = 2^q \wedge (a+b)/2 = (a+b)/2 \wedge \mathsf{min}(z_0, b - a \leq 2^{z_0 - q - 1})\right\} \\
\quad \mathtt{var}\ x := (a + b)/\iota(2)\ \mathtt{in} \\
\downarrow\left\{x : \mathsf{R} \mid a, b \in \mathbb{Q} \wedge 3 \leq a < \pi < b \leq 4 \wedge \delta = 2^q \wedge x = (a+b)/2 \wedge \mathsf{min}(z_0, b - a \leq 2^{z_0 - q - 1})\right\} \\
\quad \mathsf{bisect} \\
\downarrow\left\{a, b \in \mathbb{Q} \wedge 3 \leq a < \pi < b \leq 4 \wedge \delta = 2^q \wedge \forall z.\ (\mathsf{min}(z, b - a \leq 2^{z - q - 1})) \Rightarrow z < z_0\right\}
\end{array}$$

Consequently, we ensure that ϕ_{inv} is indeed a loop-invariant and ψ is indeed a loop-variant.

Using the *rule for while loop*, we derive the following specification:

$$\begin{array}{l}
\{a, b \in \mathbb{Q} \land 3 \le a < \pi < b \le 4 \land \delta = 2^q\} \\
\quad \mathsf{w}_{\text{out}} \\
\downarrow\{a, b \in \mathbb{Q} \land 3 \le a < \pi < b \le 4 \land \delta = 2^q \land b - a < \delta\} \\
\quad a \\
\downarrow\{y : \mathsf{R} \mid y = a \land a, b \in \mathbb{Q} \land 3 \le a < \pi < b \le 4 \land \delta = 2^q \land b - a < \delta\} \\
\downarrow\{y : \mathsf{R} \mid |y - \pi| \le 2^{-q}\}
\end{array}$$

Since the postcondition of the above specification does not refer to a, b, or δ, we derive the following specification:

$$\begin{array}{l}
\{\mathsf{True}\} \\
\{3, 4 \in \mathbb{Q} \land 3 \le 3 < \pi < 4 \le 4 \land 2^{-q} = 2^{-q}\} \\
\quad \texttt{var}\ \delta := \mathsf{prec}(-q)\ \texttt{in} \\
\downarrow\{3, 4 \in \mathbb{Q} \land 3 \le 3 < \pi < 4 \le 4 \land \delta = 2^{-q}\} \\
\quad \texttt{var}\ a := \iota(3)\ \texttt{in} \\
\downarrow\{a, 4 \in \mathbb{Q} \land 3 \le a < \pi < 4 \le 4 \land \delta = 2^{-q}\} \\
\quad \texttt{var}\ b := \iota(4)\ \texttt{in} \\
\downarrow\{a, b \in \mathbb{Q} \land 3 \le a < \pi < b \le 4 \land \delta = 2^{-q}\} \\
\quad \mathsf{w}_{\text{out}}; a \\
\downarrow\{y : \mathsf{R} \mid |y - \pi| \le 2^{-q}\}
\end{array}$$

In conclusion, we have the specification derived:

$$\{\mathsf{True}\}\ \mathsf{pi\_approx}\ \downarrow\{y : \mathsf{R} \mid |y - \pi| \le 2^{-q}\}$$

Let $\psi \coloneqq z = \pi$ and $\psi' \coloneqq |y - \pi| \le 2^{-q}$. It holds that $\exists z.\ (\forall q > 0.\ \forall y.\ |y - \pi| \le 2^{-q} \Rightarrow |y - z| \le 2^{-q}) \land z = \pi$ by simply letting $z = \pi$. Hence, the *rule for limit* yields

$$\{\mathsf{True}\}\ \texttt{lim}(q, \mathsf{pi\_approx})\ \downarrow\{y : \mathsf{R} \mid y = \pi\}$$

which confirms that the denotation of pi is indeed π.

5.6 (Relative) Completeness

We want our verification calculus to be complete in that any correct specification is derivable using the proof rules. However, we know that it is impossible as Clerical can express Peano arithmetic. Hence, the formal system is not complete anyway. However, the question remains: is the failure due to the design of the formal system, or is it due to the incompleteness of the underlying logic? The incompleteness of the underlying logic is not what we can help for. Anyhow, we need to do our best and make the design of our verification calculus not contribute to the incompleteness.

For a well-typed read-only expression $\Gamma \vdash e : \tau$, a context of auxiliary variables Ξ, and a postcondition $\Xi, \Gamma, y{:}\tau \Vdash \psi$, we say a precondition $\Xi, \Gamma \Vdash \mathsf{wp}_{\Xi,y}(\Gamma \vdash: \tau, \psi)$ a *weakest partial precondition*, and $\Xi, \Gamma \Vdash \mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma \vdash: \tau, \psi)$ a *weakest total precondition* if the following holds:

$$\begin{array}{l}
[\![\Xi, \Gamma \Vdash \mathsf{wp}_{\Xi,y}(\Gamma \vdash e : \tau, \psi)]\!] = \\
\quad \{(\xi, \gamma) \in [\![\Xi, \Gamma]\!] \mid \mathsf{e} \notin [\![\Gamma \vdash e : \tau]\!]\gamma \land \forall v \in [\![\Gamma \vdash e : \tau]\!]\gamma.\ (\xi, \gamma, y \mapsto v) \in [\![\Xi, \Gamma, y{:}\tau \Vdash \psi]\!]\}, \\
[\![\Xi, \Gamma \Vdash \mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma \vdash e : \tau, \psi)]\!] = \\
\quad \{(\xi, \gamma) \in [\![\Xi, \Gamma]\!] \mid \bot \notin [\![\Gamma \vdash e : \tau]\!]\gamma \land \forall v \in [\![\Gamma \vdash e : \tau]\!]\gamma.\ (\xi, \gamma, y \mapsto v) \in [\![\Xi, \Gamma, y{:}\tau \Vdash \psi]\!]\}\ .
\end{array}$$

In words, $[\![\mathsf{wp}_{\Xi,y}(\Gamma \vdash e : \tau, \psi)]\!]$ is the set of states which makes the evaluation of e well-defined and the results of every terminating branches satisfy ψ. And, $[\![\mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma \vdash e : \tau, \psi)]\!]$ is the set of states which makes the evaluation of e well-defined and every branches terminate resulting in ψ.

Similarly, for a well-typed read-write expression $\Gamma; \Delta \Vdash c : \tau$, a context of auxiliary variables Ξ, and a postcondition $\Xi, \Gamma, \Delta, y{:}\tau \Vdash \psi$, we say a precondition $\Xi, \Gamma, \Delta \Vdash \mathsf{wp}_{\Xi,y}(\Gamma; \Delta \Vdash c : \tau, \psi)$ a weakest partial precondition and $\Xi, \Gamma, \Delta \Vdash \mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma; \Delta \Vdash c : \tau, \psi)$ a weakest partial precondition if the following holds:

$$
\begin{aligned}
&[\![\Xi, \Gamma, \Delta \Vdash \mathsf{wp}_{\Xi,y}(\Gamma; \Delta \Vdash c : \tau, \psi)]\!] = \\
&\quad \{(\xi, \gamma, \delta) \in [\![\Xi, \Gamma, \Delta]\!] \mid \mathsf{e} \notin [\![\Gamma; \Delta \Vdash c : \tau]\!]\gamma\, \delta \wedge \\
&\qquad\qquad \forall (\delta', v) \in [\![\Gamma; \Delta \Vdash c : \tau]\!]\gamma\, \delta.\ (\xi, \gamma, \delta, y \mapsto v) \in [\![\Xi, \Gamma, \Delta, y{:}\tau \Vdash \psi]\!]\}, \\
&[\![\Xi, \Gamma, \Delta \Vdash \mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma \vdash e : \tau, \psi)]\!] = \\
&\quad \{(\xi, \gamma, \delta) \in [\![\Xi, \Gamma, \Delta]\!] \mid \bot \notin [\![\Gamma; \Delta \Vdash c : \tau]\!]\gamma\, \delta \wedge \\
&\qquad\qquad \forall (\delta', v) \in [\![\Gamma; \Delta \Vdash c : \tau]\!]\gamma\, \delta.\ (\xi, \gamma, \delta, y \mapsto v) \in [\![\Xi, \Gamma, \Delta, y{:}\tau \Vdash \psi]\!]\}\ .
\end{aligned}
$$

Note that though the sets always exist, due to the expressiveness of the assertion language, they may not be definable. In order to focus purely on the verification calculus, we pose two assumptions on our assertion language. An assertion language is *expressive complete* if for any Γ and $S \subseteq [\![\Gamma]\!]$, there is $\Gamma \Vdash \phi$ such that $S = [\![\Gamma \Vdash \phi]\!]$. And, an assertion language is *semantic complete* if for any $\Gamma \Vdash \phi$, $\Gamma \vdash \phi$ if and only if $[\![\Gamma \Vdash \phi]\!] = [\![\Gamma]\!]$.

Theorem 5.2. The proof rules of Clerical is complete assuming the assertion language is expressive and semantic complete.

Proof. Since, we assumed expressive completeness, all weakest (partial) preconditions exist.

Consider any correct specifications:

$$\Xi \triangleright \Gamma \vdash \{\phi^?\}\ \ e\ \ ?\{y : \tau \mid \psi\} \quad \Xi \triangleright \Gamma; \Delta \Vdash \{\phi^?\}\ \ c\ \ ?\{y : \tau \mid \psi\},$$

We first show that the specifications

$$\Xi \triangleright \Gamma \vdash \{\mathsf{wp}^?_{\Xi,y}(\Gamma \vdash c : \tau, \psi)\}\ \ e\ \ ?\{y : \tau \mid \psi\} \quad \Xi \triangleright \Gamma; \Delta \Vdash \{\mathsf{wp}^?_{\Xi,y}(\Gamma; \Delta \Vdash c : \tau, \psi)\}\ \ c\ \ ?\{y : \tau \mid \psi\},$$

are derivable. Since we have assumed that the assertion language is semantic complete, it holds that

$$\Xi, \Gamma \vdash \phi^? \Rightarrow \mathsf{wp}^?_{\Xi,y}(\Gamma \vdash c : \tau, \psi) \quad \text{and} \quad \Xi, \Gamma, \Delta \vdash \phi^? \Rightarrow \mathsf{wp}^?_{\Xi,y}(\Gamma; \Delta \Vdash c : \tau, \psi).$$

Hence, by using the rule for precondition strengthening, we get the desired specifications derived.

We prove it by induction on the well-typedness of e and c.

• The case when the well-typedness is $\Gamma \vdash \mathtt{true} : \mathsf{B}$:

Suppose any correct specification:

$$\Xi \triangleright \Gamma \vdash \{\phi\}\ \ \mathtt{true}\ \ ?\{y : \mathsf{B} \mid \psi\}$$

From the *rule for constants* and the *rule for partial correctness from total correctness*, we can derive

$$\Xi \triangleright \Gamma \vdash \{\psi[\mathtt{true}/y]\}\ \ \mathtt{true}\ \ ?\{y : \mathsf{B} \mid \psi\}$$

Suppose any $(\xi, \gamma) \in [\![\Xi, \Gamma \Vdash \phi]\!]$. By the assumption, it holds that $(\xi, \gamma, y \mapsto tt) \in [\![\Xi, \Gamma, y{:}\mathsf{B} \Vdash \psi]\!]$. Therefore, $(\xi, \gamma) \in [\![\Xi, \Gamma \Vdash \psi[\mathtt{true}/y]]\!]$ holds, which implies $\Xi, \Gamma \vdash \phi \Rightarrow \psi[\mathtt{true}/y]$. Hence, by the *rule for precondition strengthening*, we have

$$\Xi \triangleright \Gamma \vdash \{\phi\}\ \ \mathtt{true}\ \ ?\{y : \mathsf{B} \mid \psi\}$$

• The case when the well-typedness is $\Gamma \vdash \texttt{false} : \mathsf{B}$, $\Gamma \vdash k : \mathsf{Z}$, $\Gamma \vdash \texttt{skip} : \mathsf{U}$, or $\Gamma \vdash x : \tau$, it can be done identically to above.

• The case when the well-typedness is $\Gamma \vdash \iota(e) : \mathsf{R}$:

From the induction hypothesis, we can derive

$$\Xi \triangleright \Gamma \vdash \left\{\mathsf{wp}^{?}_{\Xi,y}(\Gamma \vdash e : \mathsf{Z}, \exists z : \mathsf{R}\,.\,\psi \wedge z = \iota(y))\right\}\ e : \mathsf{Z}\ \ ?\left\{y : \mathsf{Z} \mid \exists z : \mathsf{R}\,.\,\psi \wedge z = \iota(y)\right\}$$

Hence, applying the *rule for coercion*, we derive the specification:

$$\Xi \triangleright \Gamma \vdash \left\{\mathsf{wp}^{?}_{\Xi,y}(\Gamma \vdash e : \mathsf{Z}, \exists z : \mathsf{R}\,.\,\psi \wedge z = \iota(y))\right\}\ \iota(e)\ \ ?\left\{z : \mathsf{R} \mid \exists y : \mathsf{R}\,.\,\exists z : \mathsf{R}\,.\,\psi \wedge z = \iota(y) \wedge z = \iota(y)\right\}$$

As $\psi \Rightarrow \exists z : \mathsf{R}\,.\,\psi \wedge z = \iota(y)$ and $\exists y : \mathsf{R}\,.\,\exists z : \mathsf{R}\,.\,\psi \wedge z = \iota(y) \wedge z = \iota(y) \Rightarrow \psi$, we have the desired specification.

• The case when the well-typedness is $\Gamma \vdash e_1 \odot e_2 : \mathsf{Z}$ for $\odot \in \{+, -, \times\}$:

Suppose any state $(\xi, \gamma) \in [\![\mathsf{wp}^{?}_{\Xi,y}(\Gamma \vdash e_1 \odot e_2 : \mathsf{Z}, \psi)]\!]$. Then, clearly, it holds that

$$(\xi, \gamma) \in [\![\mathsf{wp}^{?}_{\Xi,y}(\Gamma \vdash e_1 : \mathsf{Z}, \mathsf{True}) \wedge \mathsf{wp}^{?}_{\Xi,y}(\Gamma \vdash e_2 : \mathsf{Z}, \mathsf{True})]\!].$$

And, using the induction hypothesis, we can derive the following specifications:

$$\begin{array}{r}
\Xi \triangleright \Gamma \vdash \left\{\mathsf{wp}^{?}_{\Xi,y}(\Gamma \vdash e_1 \odot e_2 : \mathsf{Z}, \psi)\right\}\ e_1\ \ ?\left\{y : \mathsf{Z} \mid \mathsf{True}\right\} \\
\Xi \triangleright \Gamma \vdash \left\{\mathsf{wp}^{?}_{\Xi,y}(\Gamma \vdash e_1 \odot e_2 : \mathsf{Z}, \psi)\right\}\ e_2\ \ ?\left\{y : \mathsf{Z} \mid \mathsf{True}\right\} \\
\Xi, x_1{:}\mathsf{Z} \triangleright \Gamma \vdash \left\{\mathsf{wp}_{(\Xi,x_1:\mathsf{Z}),y}(\Gamma \vdash e_1 : \mathsf{Z}, y \neq x_1)\right\}\ e_1\ \left\{y : \mathsf{Z} \mid y \neq x_1\right\} \\
\Xi, x_2{:}\mathsf{Z} \triangleright \Gamma \vdash \left\{\mathsf{wp}_{(\Xi,x_2:\mathsf{Z}),y}(\Gamma \vdash e_2 : \mathsf{Z}, y \neq x_2)\right\}\ e_2\ \left\{y : \mathsf{Z} \mid y \neq x_2\right\}
\end{array}$$

For any $v_1 \in \mathbb{Z}$, if $(\xi, x_1 \mapsto v_1, \gamma) \in [\![\neg \mathsf{wp}_{\Xi,x_1:\mathsf{Z}}(\Gamma \vdash e_1 : \mathsf{Z}, y \neq x_1)]\!]$ holds, $v_1 \in [\![\Gamma \vdash e_1 : \mathsf{Z}]\!]\gamma$ holds. Therefore, for any $v_1, v_2 \in \mathbb{Z}$ such that

$$\begin{array}{l}
(\xi, x_1 \mapsto v_1, x_2 \mapsto v_2, \gamma) \in \\
[\![\mathsf{wp}^{?}_{\Xi,y}(\Gamma \vdash e_1 \odot e_2 : \mathsf{Z}, \psi) \wedge \neg \mathsf{wp}_{\Xi,x_1:\mathsf{Z}}(\Gamma \vdash e_1 : \mathsf{Z}, y \neq x_1) \wedge \neg \mathsf{wp}_{\Xi,x_2:\mathsf{Z}}(\Gamma \vdash e_2 : \mathsf{Z}, y \neq x_2)]\!],
\end{array}$$

it holds that $v_1 \in [\![\Gamma \vdash e_1 : \mathsf{Z}]\!]\gamma$ and $v_2 \in [\![\Gamma \vdash e_2 : \mathsf{Z}]\!]\gamma$. By the assumption, we have $(\xi, x_1 \mapsto v_1, x_2 \mapsto v_2, \gamma) \in [\![\Xi, x_1{:}\mathsf{Z}, x_2{:}\mathsf{Z}, \Gamma \Vdash \psi[(x_1 \odot x_2)/y]]\!]$. Therefore, the side-condition of *the rules for integer arithmetic* is provable (by the completeness). Hence, using the rule, we get the desired specification.

• The case when the well-typedness is $\Gamma \vdash e_1 \boxdot e_2 : \mathsf{R}$ for $\boxdot \in \{+, -, \times\}$, $\Gamma \vdash e_1 < e_2 : \mathsf{B}$, or $\Gamma \vdash e_1 = e_2 : \mathsf{B}$, it can be done very similarly to above.

• The case when the well-typedness is $\Gamma \vdash e_1 \hat{<} e_2 : \mathsf{B}$:

The partial correctness case can be done in a very similar way to the case of integer arithmetic.

Suppose any state (ξ, γ) in $[\![\Xi, \Gamma \Vdash \mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma \vdash e_1 \hat{<} e_2 : \mathsf{B}, \psi)]\!]$. Then, with the same argument used in the case of integer arithmetic, it holds that $(\xi, \gamma) \in [\![\Xi, \Gamma \Vdash \mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma \vdash e_1 : \mathsf{R}, \mathsf{True})]\!]$ and $(\xi, \gamma) \in [\![\Xi, \Gamma \Vdash \mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma \vdash e_2 : \mathsf{R}, \mathsf{True})]\!]$.

And, the induction hypothesis ensures that the following specifications are derivable:

$$\begin{array}{r}
\Xi \triangleright \Gamma \vdash \left\{\mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma \vdash e_1 : \mathsf{R}, \mathsf{True})\right\}\ e_1\ \ \downarrow\left\{y : \mathsf{R} \mid \mathsf{True}\right\} \\
\Xi \triangleright \Gamma \vdash \left\{\mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma \vdash e_2 : \mathsf{R}, \mathsf{True})\right\}\ e_2\ \ \downarrow\left\{y : \mathsf{R} \mid \mathsf{True}\right\} \\
\Xi, x_1{:}\mathsf{R} \triangleright \Gamma \vdash \left\{\mathsf{wp}_{(\Xi,x_1:\mathsf{R}),y}(\Gamma \vdash e_1 : \mathsf{R}, y \neq x_1)\right\}\ e_1\ \left\{y : \mathsf{R} \mid y \neq x_1\right\} \\
\Xi, x_2{:}\mathsf{R} \triangleright \Gamma \vdash \left\{\mathsf{wp}_{(\Xi,x_2:\mathsf{R}),y}(\Gamma \vdash e_2 : \mathsf{R}, y \neq x_2)\right\}\ e_2\ \left\{y : \mathsf{R} \mid y \neq x_2\right\}
\end{array}$$

Suppose any $v_1, v_2 \in \mathbb{R}$ such that

$$(\xi, x_1 \mapsto v_1, x_2 \mapsto v_2, \gamma) \in$$
$$[\![\Xi, \Gamma \Vdash \mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma \vdash e_1 \hat{<} e_2 : \mathsf{B}, \psi) \wedge \neg \mathsf{wp}_{\Xi,x_1:\mathsf{R},y}(\Gamma \vdash e_1 : \mathsf{R}, y \neq x_1) \wedge \neg \mathsf{wp}_{\Xi,x_2:\mathsf{R},y}(\Gamma \vdash e_2 : \mathsf{R}, y \neq x_2)]\!].$$

Then, it holds that $v_1 \in [\![\Gamma \vdash e_1 : \mathsf{R}]\!]\gamma$ and $v_2 \in [\![\Gamma \vdash e_2 : \mathsf{R}]\!]\gamma$. If $v_1 < v_2$, it holds that $tt \in [\![\Gamma \vdash e_1 \hat{<} e_2 : \mathsf{R}]\!]\gamma$ and if $v_1 > v_2$, it holds that $ff \in [\![\Gamma \vdash e_1 \hat{<} e_2 : \mathsf{R}]\!]\gamma$. However, $v_1 = v_2$ does not hold, since $(\xi, x_1 \mapsto v_1, x_2 \mapsto v_2, \gamma) \in [\![\Xi, \Gamma \Vdash \mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma \vdash e_1 \hat{<} e_2 : \mathsf{B}, \psi)]\!]$. Therefore, the side-condition holds. And, using the *total correctness rule for real comparison*, we get the desired specification.

- The case when the well-typedness is $\Gamma \vdash e^{-1} : \mathsf{R}$ is done very similarly.

- The case when the well-typedness is $\Gamma \vdash \mathtt{lim}\, x\,.\, e : \mathsf{R}$:
  See that

$$\mathsf{wp}^{\downarrow}_{\Xi,z}(\Gamma \vdash \mathtt{lim}\, x\,.\, e : \mathsf{R}, \psi) \Rightarrow \exists z : \mathsf{R}\,.\, \psi \wedge \forall x : \mathsf{Z}\,.\, x \geq 0 \Rightarrow \mathsf{wp}^{\downarrow}_{(\Xi,z:\mathsf{R}),y}(\Gamma, x{:}\mathsf{Z} \vdash e : \mathsf{R}, |y - z| \leq 2^{-x}).$$

By the induction hypothesis, we can derive the specification:

$$\Xi, z{:}\mathsf{R} \triangleright \Gamma, x{:}\mathsf{Z} \vdash \left\{\mathsf{wp}^{\downarrow}_{(\Xi,z:\mathsf{R}),y}(\Gamma, x{:}\mathsf{Z} \vdash e : \mathsf{R}, |y - z| \leq 2^{-x})\right\}\ e\ \ \downarrow\!\left\{y : \mathsf{R} \mid |y - z| \leq 2^{-x}\right\}$$

See that the side-condition becomes

$$\begin{aligned}&\left(\exists z{:}\mathsf{R}\,.\, \psi \wedge \forall x{:}\mathsf{Z}\,.\, x \geq 0 \Rightarrow \mathsf{wp}^{\downarrow}_{(\Xi,z:\mathsf{R}),y}(\Gamma, x : \mathsf{Z} \vdash e : \mathsf{R}, |y - z| \leq 2^{-x})\right) \Rightarrow \\ &\quad \exists z{:}\mathsf{R}\,.\, \left(\forall x{:}\mathsf{Z}\,.\, x \geq 0 \Rightarrow (\mathsf{wp}^{\downarrow}_{(\Xi,z:\mathsf{R}),y}(\Gamma, x{:}\mathsf{Z} \vdash e{:}\mathsf{R}, |y - z| \leq 2^{-x}) \wedge (\forall y{:}\mathsf{R}\,.\, |y - z| \leq 2^{-x} \rightarrow |y - z| \leq 2^{-z}))\right) \\ &\quad \wedge \psi\end{aligned}$$

which is trivially provable. The *rule for limit* derives the desired specification.

- The case when the well-typedness is $\Gamma \vdash c_1; c_2 : \tau$:
  See that

$$\mathsf{wp}^{?}_{\Xi,y}(\Gamma; \Delta \Vdash c_1; c_2 : \tau, \psi) \Rightarrow \mathsf{wp}^{?}_{\Xi,y}(\Gamma; \Delta \Vdash c_1 : \mathsf{U}, \mathsf{wp}^{?}_{\Xi,y}(\Gamma; \Delta \Vdash c_2, \psi))$$

By the induction hypothesis, we can derive the specification:

$$\begin{aligned}&\Xi \triangleright \Gamma; \Delta \Vdash \left\{\mathsf{wp}^{?}_{\Xi,y}(\Gamma; \Delta \Vdash c_2, \psi)\right\}\ c_2\ \ ?\!\left\{y : \tau \mid \psi\right\} \\ &\Xi \triangleright \Gamma; \Delta \Vdash \left\{\mathsf{wp}^{?}_{\Xi,y}(\Gamma; \Delta \Vdash c_1 : \mathsf{U}, \mathsf{wp}^{?}_{\Xi,y}(\Gamma; \Delta \Vdash c_2, \psi))\right\}\ c_1\ \ ?\!\left\{y : \mathsf{U} \mid \mathsf{wp}^{?}_{\Xi,y}(\Gamma; \Delta \Vdash c_2, \psi)\right\}\end{aligned}$$

Since, y does not appear free in $\mathsf{wp}^{?}_{\Xi,y}(\Gamma; \Delta \Vdash c_2, \psi)$, we can apply the rules for sequencing to get

$$\Xi \triangleright \Gamma; \Delta \Vdash \left\{\mathsf{wp}^{?}_{\Xi,y}(\Gamma; \Delta \Vdash c_1; c_2 : \tau, \psi)\right\}\ c_1; c_2\ \ ?\!\left\{y : \tau \mid \psi\right\}$$

- The case when the well-typedness is $\Gamma; \Delta \Vdash \mathtt{var}\ x := e\ \mathtt{in}\ c : \tau$:
  The induction hypothesis says the specifications are derivable:

$$\begin{aligned}&\Xi \triangleright \Gamma; \Delta, x{:}\sigma \Vdash \left\{\mathsf{wp}^{?}_{\Xi,y}(\Gamma; \Delta, x{:}\sigma \Vdash c : \tau, \psi)\right\}\ c\ \ ?\!\left\{\psi\right\} \\ &\Xi \triangleright \Gamma; \Delta \Vdash \left\{\mathsf{wp}^{?}_{\Xi,y}(\Gamma, \Delta \vdash e : \sigma, \mathsf{wp}^{?}_{\Xi,y}(\Gamma; \Delta, x{:}\sigma \Vdash c{:}\tau, \psi)[y/x])\right\}\ e\ ?\!\left\{y{:}\tau \mid \mathsf{wp}^{?}_{\Xi,y}(\Gamma; \Delta, x{:}\sigma \Vdash c{:}\tau, \psi)[y/x]\right\}\end{aligned}$$

Hence, *the rule for local variable* yields

$$\Xi \triangleright \Gamma; \Delta \Vdash \left\{\mathsf{wp}^{?}_{\Xi,y}(\Gamma, \Delta \vdash e : \sigma, \mathsf{wp}^{?}_{\Xi,y}(\Gamma; \Delta, x{:}\sigma \Vdash c : \tau, \psi)[y/x])\right\}\ \mathtt{var}\ x := e\ \mathtt{in}\ c\ \ ?\!\left\{y : \tau \mid \exists x.\ \psi\right\}$$

Since x does not appear free in ψ, using the implication

$$\mathsf{wp}^{?}_{\Xi,y}(\Gamma;\Delta \Vdash \texttt{var}\ x := e\ \texttt{in}\ c : \tau, \psi) \Rightarrow \mathsf{wp}^{?}_{\Xi,y}(\Gamma, \Delta \vdash e : \delta, \mathsf{wp}^{?}_{\Xi,y}(\Gamma, \Delta, x{:}\sigma \Vdash c : \tau, \psi)[y/x])$$

we can derive the desired spcification:

$$\Xi \triangleright \Gamma; \Delta \Vdash \left\{\mathsf{wp}^{?}_{\Xi,y}(\Gamma;\Delta \Vdash \texttt{var}\ x := e\ \texttt{in}\ c : \tau, \psi)\right\}\ \texttt{var}\ x := e\ \texttt{in}\ c\ \ ?\left\{y : \tau \mid \psi\right\}$$

• The case when the well-typedness is $\Gamma; \Delta \Vdash x := e : \mathsf{U}$:

The induction hypothesis derives the specification:

$$\Xi \triangleright \Gamma, \Delta \vdash \left\{\mathsf{wp}^{?}_{\Xi,y}(\Gamma, \Delta \vdash e : \tau, \psi[y/x])\right\}\ e\ \ ?\left\{y : \tau \mid \psi[y/x]\right\}$$

Letting $\theta \coloneqq \psi$ in *the rule for assignment* derives

$$\Xi \triangleright \Gamma; \Delta \Vdash \left\{\mathsf{wp}^{?}_{\Xi,y}(;\Gamma, \Delta \vdash e : \tau, \psi[y/x]) \wedge \forall y.\ (\psi[y/x] \to \psi[y/x])\right\}\ x := e\ \ ?\left\{\psi\right\}\ .$$

Since the implication

$$\mathsf{wp}^{?}_{\Xi,\_}(\Gamma;\Delta \Vdash x := e : \mathsf{U}, \psi) \Rightarrow \mathsf{wp}^{?}_{\Xi,y}(\Gamma, \Delta \vdash e : \tau, \psi[y/x])$$

and $\mathsf{wp}^{?}_{\Xi,y}(\Gamma, \Delta \vdash e : \tau, \psi[y/x]) \Rightarrow \mathsf{wp}^{?}_{\Xi,y}(\Gamma, \Delta \vdash e : \tau, \psi[y/x]) \wedge \forall y.\ (\psi[y/x] \Rightarrow \psi[y/x])$ hold, using the *rule of precondition weakening*, we get the following desired specification:

$$\Xi \triangleright \Gamma; \Delta \Vdash \left\{\mathsf{wp}^{?}_{\Xi,\_}(\Gamma;\Delta \Vdash x := e : \mathsf{U}, \psi)\right\}\ x := e\ \ ?\left\{\psi\right\}$$

This can be done very similarly to the case of guarded cases.

• The partial correctness of the case when the well-typedness is $\Gamma; \Delta \Vdash \texttt{case}\ e_1 \Rightarrow c_1 \mid \cdots \mid e_n \Rightarrow c_n\ \texttt{end} : \tau$

Suppose any $(\xi, \gamma, \delta) \in [\![\Xi, \Gamma, \Delta \Vdash \mathsf{wp}_{\Xi,y}(\Gamma;\Delta \Vdash \texttt{case}\ e_1 \Rightarrow c_1 \mid \cdots \mid e_n \Rightarrow c_n : \tau, \psi)]\!]$. In order to exclude e in the denotation, it holds that $\forall i.\ \mathsf{e} \notin [\![\Gamma, \Delta \vdash e_i : \mathsf{B}]\!](\gamma, \delta)$. Hence, (ξ, γ, δ) is in $[\![\Xi, \Gamma, \Delta \Vdash \mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e_i, \mathsf{True})]\!]$ for all i.

Suppose (ξ, γ, δ) is in $[\![\Xi, \Gamma, \Delta \Vdash \neg\mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e_n : \mathsf{B}, y = \texttt{false})]\!]$. Then, $tt \in [\![\Gamma, \Delta \vdash e_i : \mathsf{B}]\!](\gamma, \delta)$ holds. Hence, in order to make $\mathsf{e} \notin [\![\Gamma; \Delta \Vdash c_i : \tau]\!]\gamma\,\delta$ and any $(\delta', v) \in [\![\Gamma; \Delta \Vdash c_i : \tau]\!]$ satisfy ψ with ξ, γ, the state (ξ, γ, δ) is in $[\![\Xi, \Gamma, \Delta \Vdash \mathsf{wp}_{\Xi,y}(\Gamma;\Delta \Vdash c_1 : \tau, \psi)]\!]$.

Hence, the implication holds:

$$\begin{aligned}
&\mathsf{wp}_{\Xi,y}(\Gamma;\Delta \Vdash \texttt{case}\ e_1 \Rightarrow c_1 \mid \cdots \mid e_n \Rightarrow c_n : \tau, \psi) \Rightarrow \\
&\quad \mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e_1, \mathsf{True}) \wedge \cdots \wedge \mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e_n, \mathsf{True}) \\
&\wedge (\neg\mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e_1 : \mathsf{B}, y = \texttt{false}) \Rightarrow \mathsf{wp}_{\Xi,y}(\Gamma;\Delta \Vdash c_1 : \tau, \psi)) \\
&\ \vdots \\
&\wedge (\neg\mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e_n : \mathsf{B}, y = \texttt{false}) \Rightarrow \mathsf{wp}_{\Xi,y}(\Gamma;\Delta \Vdash c_n : \tau, \psi))
\end{aligned}$$

Let us write Φ to denote the right-hand-side of the above implication.

By the induction hypothesis, we have the specifications derived:

$$\begin{aligned}
&\Xi \triangleright \Gamma, \Delta \vdash \left\{\mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e_i, \mathsf{True})\right\}\ e_i\ \left\{y : \mathsf{B} \mid \mathsf{True}\right\} \\
&\Xi \triangleright \Gamma, \Delta \vdash \left\{\mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e_i : \mathsf{B}, y = \texttt{false})\right\}\ e_i\ \left\{y : \mathsf{B} \mid y = \texttt{false}\right\} \\
&\Xi \triangleright \Gamma; \Delta \Vdash \left\{\mathsf{wp}_{\Xi,y}(\Gamma;\Delta \Vdash c_i : \tau, \psi)\right\}\ c_i\ \left\{y : \tau \mid \psi\right\}
\end{aligned}$$

See that $\Phi \Rightarrow \mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e_i, \mathsf{True})$ and $\Phi \wedge \neg \mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e_i : \mathsf{B}, y = \mathtt{false}) \Rightarrow \mathsf{wp}_{\Xi,y}(\Gamma; \Delta \Vdash c_i : \tau, \psi)$ hold for all i. Hence, applying *the rule for precondition strengthening*, we get

$$\Xi \triangleright \Gamma, \Delta \vdash \left\{\mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e_i, \mathsf{True})\right\}\ e_i\ \left\{y : \mathsf{B} \mid \mathsf{True}\right\}$$
$$\Xi \triangleright \Gamma; \Delta \Vdash \left\{\Phi \wedge \neg \mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e_i : \mathsf{B}, y = \mathtt{false})\right\}\ c_i\ \left\{y : \tau \mid \psi\right\}$$

Applying *the rule for guarded cases* and *the rule for precondition strengthening*, we get the desired specification derived.

• The total correctness of the case when the well-typedness is $\Gamma; \Delta \Vdash \mathtt{case}\ e_1 \Rightarrow c_1 \mid \cdots \mid e_n \Rightarrow c_n\ \mathtt{end} : \tau$

Suppose $(\xi, \gamma, \delta) \in [\![\Xi, \Gamma, \Delta \Vdash \mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma; \Delta \Vdash \mathtt{case}\ e_1 \Rightarrow c_1 \mid \cdots \mid e_n \Rightarrow c_n : \tau, \psi)]\!]$. Then, in order to ensure $\mathsf{e} \notin [\![\Gamma, \Delta \vdash e_i : \mathsf{B}]\!](\gamma, \delta)$ for all i, $(\xi, \gamma, \delta) \in [\![\Xi, \Gamma, \Delta \Vdash \mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e_i, \mathsf{True})]\!]$ holds for all i.

Note that if there is no i where $\{tt\} = [\![\Gamma, \Delta \vdash e_i : \mathsf{B}]\!](\gamma, \delta)$ holds, $\bot$ is in the denotation under γ and δ. Hence, there is i where $(\xi, \gamma, \delta) \in [\![\Xi, \Gamma, \Delta \Vdash \mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma, \Delta \vdash e_i, y = \mathtt{true})]\!]$ holds.

Therefore, we have the implication:

$$\begin{aligned}
&\mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma; \Delta \Vdash \mathtt{case}\ e_1 \Rightarrow c_1 \mid \cdots \mid e_n \Rightarrow c_n : \tau, \psi) \Rightarrow \\
&\quad \mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e_1, \mathsf{True}) \wedge \cdots \wedge \mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e_n, \mathsf{True}) \\
&\wedge \left(\mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma, \Delta \vdash e_1, y = \mathtt{true}) \vee \cdots \vee \mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma, \Delta \vdash e_n, y = \mathtt{true})\right) \\
&\wedge (\neg \mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e_1 : \mathsf{B}, y = \mathtt{false}) \Rightarrow \mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma; \Delta \Vdash c_1 : \tau, \psi)) \\
&\ \vdots \\
&\wedge (\neg \mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e_n : \mathsf{B}, y = \mathtt{false}) \Rightarrow \mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma; \Delta \Vdash c_n : \tau, \psi))
\end{aligned}$$

Let us write Φ to denote the right-hand-side of the above implication.

By the induction hypothesis, we have the specifications derived:

$$\Xi \triangleright \Gamma, \Delta \vdash \left\{\mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e_i, \mathsf{True})\right\}\ e_i\ \left\{y : \mathsf{B} \mid \mathsf{True}\right\}$$
$$\Xi \triangleright \Gamma, \Delta \vdash \left\{\mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma, \Delta \vdash e_i, y = \mathtt{true})\right\}\ e_i\ \downarrow\left\{y : \mathsf{B} \mid y = \mathtt{true}\right\}$$
$$\Xi \triangleright \Gamma, \Delta \vdash \left\{\mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e_i : \mathsf{B}, y = \mathtt{false})\right\}\ e_i\ \left\{y : \mathsf{B} \mid y = \mathtt{false}\right\}$$
$$\Xi \triangleright \Gamma; \Delta \Vdash \left\{\mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma; \Delta \Vdash c_i : \tau, \psi)\right\}\ c_i\ \downarrow\left\{y : \tau \mid \psi\right\}$$

Let $\phi_i \coloneqq \mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma, \Delta \vdash e_i, y = \mathtt{true})$ and $\theta_i \coloneqq \mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e_i : \mathsf{B}, y = \mathtt{false})$. Note that $\Phi \Rightarrow \mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e_i, \mathsf{True})$ holds and for any i, and $(\Phi \wedge (\phi_1 \vee \cdots \vee \phi_n) \wedge \neg\theta_i) \Rightarrow \mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma; \Delta \Vdash c_i : \tau, \psi)$ holds.

Therefore, by applying *the rule for precondition strengthening* and *the total correctness rule for guarded cases*, we get

$$\Xi \triangleright \Gamma, \Delta \Vdash \left\{\Phi \wedge (\phi_1 \vee \cdots \vee \phi_n)\right\}\ \mathtt{case}\ e_1 \Rightarrow c_1 \mid \cdots \mid e_n \Rightarrow c_n\ \downarrow\left\{y : \tau \mid \psi\right\}$$

As $\mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma; \Delta \Vdash \mathtt{case}\ e_1 \Rightarrow c_1 \mid \cdots \mid e_n \Rightarrow c_n : \tau, \psi) \Rightarrow \Phi$ and $\Phi \Rightarrow \Phi \wedge (\phi_1 \vee \cdots \vee \phi_n)$, by applying *the rule for precondition strengthening*, we get the desired specification.

• The case when the well-typedness is $\Gamma; \Delta \Vdash \mathtt{if}\ e\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2\ \mathtt{end} : \tau$ is done very similarly.

• The partial correctness of the case when the well-typedness is $\Gamma; \Delta \Vdash \mathtt{while}\ e\ \mathtt{do}\ c\ \mathtt{end} : \mathsf{U}$:

Define the sequence of formulae:

$$\begin{aligned}
\phi_0 &\coloneqq \mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e : \mathsf{B}, \mathsf{True}) \\
\phi_{n+1} &\coloneqq (\neg \mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e : \mathsf{B}, y = \mathtt{false}) \Rightarrow \mathsf{wp}_{\Xi,y}(\Gamma; \Delta \Vdash c : \tau, \phi_n)) \\
&\quad \wedge (\neg \mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e : \mathsf{B}, y = \mathtt{true}) \Rightarrow \psi)
\end{aligned}$$

Let us abbreviate W for $\mathsf{wp}_{\Xi,y}(\Gamma;\Delta \Vdash \texttt{while}\ e\ \texttt{do}\ c\ \texttt{end} : \mathsf{U}, \psi)$ and see that $[\![\mathsf{W}]\!] = \bigcap_{n=0}^{\infty} [\![\phi_n]\!]$.

Let W be a loop-invariant. By $\mathsf{W} \Rightarrow \mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e : \mathsf{B}, \mathsf{True})$, we have

$$\Xi \triangleright \Gamma, \Delta \vdash \left\{\mathsf{W}\right\}\ e\ \left\{y : \mathsf{B} \mid \mathsf{True}\right\}.$$

Let $\phi_{\mathrm{false}} \coloneqq \mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e : \mathsf{B}, y = \texttt{false})$, and $\phi_{\mathrm{true}} \coloneqq \mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e : \mathsf{B}, y = \texttt{true})$. See that $\neg\phi_{\mathrm{false}} \wedge \mathsf{W} \Rightarrow \mathsf{wp}_{\Xi,y}(\Gamma; \Delta \Vdash c : \tau, \phi_n)$ holds for all $n \in \mathbb{N}$. Hence, using the completeness assumption, we have the derivation:

$$\begin{array}{rcl}
\neg\phi_{\mathrm{false}} \wedge \mathsf{W} & \Rightarrow & \mathsf{wp}_{\Xi,y}(\Gamma; \Delta \Vdash c : \tau, \phi_n) \text{ for all } n \in \mathbb{N} \\
[\![\neg\phi_{\mathrm{false}} \wedge \mathsf{W}]\!] & \subseteq & [\![\mathsf{wp}_{\Xi,y}(\Gamma; \Delta \Vdash c : \tau, \phi_n)]\!] \text{ for all } n \in \mathbb{N} \\
[\![\neg\phi_{\mathrm{false}} \wedge \mathsf{W}]\!] & \subseteq & \bigcap_{n\in\mathbb{N}} [\![\mathsf{wp}_{\Xi,y}(\Gamma; \Delta \Vdash c : \tau, \phi_n)]\!] \\
 & = & [\![\mathsf{wp}_{\Xi,y}(\Gamma; \Delta \Vdash c : \tau, \mathsf{W})]\!] \\
\neg\phi_{\mathrm{false}} \wedge \mathsf{W} & \Rightarrow & \mathsf{wp}_{\Xi,y}(\Gamma; \Delta \Vdash c : \tau, \mathsf{W}).
\end{array}$$

By the implication, we have the specification

$$\Xi \triangleright \Gamma; \Delta \Vdash \left\{\neg\phi_{\mathrm{false}} \wedge \mathsf{W}\right\}\ c\ \left\{\mathsf{W}\right\}.$$

Hence, using *the rule for loop*, we the desired partial correctness specification derived:

$$\Xi \triangleright \Gamma; \Delta \Vdash \left\{\mathsf{W}\right\}\ \texttt{while}\ e\ \texttt{do}\ c\ \texttt{end}\ \left\{\mathsf{W} \wedge \neg\phi_{\mathrm{true}}\right\}.$$

Since $\mathsf{W} \wedge \neg\phi_{\mathrm{true}} \Rightarrow \psi$, we have the desired specification derived.

• The total correctness of the case when the well-typedness is $\Gamma; \Delta \Vdash \texttt{while}\ e\ \texttt{do}\ c\ \texttt{end} : \mathsf{U}$:

Define the sequence of formulae:

$$\begin{array}{rcl}
\phi_0 & \coloneqq & (\mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma, \Delta \vdash e : \mathsf{B}, y = \texttt{false})) \wedge \psi \\
\phi_{n+1} & \coloneqq & (\neg\mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e : \mathsf{B}, y = \texttt{false}) \Rightarrow \mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma; \Delta \Vdash c : \mathsf{U}, \phi_n)) \\
 & & \wedge \mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma, \Delta \vdash e : \mathsf{B}, \mathsf{True}) \\
 & & \wedge (\neg\mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e : \mathsf{B}, y = \texttt{true}) \Rightarrow \psi)
\end{array}$$

Let us abbreviate W for $\mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma; \Delta \Vdash \texttt{while}\ e\ \texttt{do}\ c\ \texttt{end} : \mathsf{U}, \psi)$ and see that $[\![\mathsf{W}]\!] = \bigcup_{n=0}^{\infty} [\![\phi_n]\!]$ holds.

Let $\Xi, \Gamma, \Delta, z{:}\mathsf{Z} \Vdash \psi$ be a formula that defines the set $[\![\psi]\!]$ such that $(\xi, \gamma, \delta, z \mapsto v) \in [\![\psi]\!]$ if and only if $(\xi, \gamma, \delta) \in [\![\phi_{v+1}]\!]$ and for any $v' < v$, $(\xi, \gamma, \delta) \in [\![\neg\phi_{v'+1}]\!]$. In other words, z satisfies ψ if and only if z is the smallest integer that satisfies ϕ_{z+1}.

Since $\mathsf{W} \Rightarrow \mathsf{wp}^{\downarrow}_{\Xi,y}(e : \mathsf{B}, \mathsf{True})$ holds, we have $\Xi \triangleright \Gamma, \Delta \vdash \left\{\mathsf{W}\right\}\ e\ \ \downarrow\left\{y : \mathsf{B} \mid \mathsf{True}\right\}$.

Similarly to the case of partial correctness, let $\phi_{\mathrm{false}} \coloneqq \mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e : \mathsf{B}, y = \texttt{false})$, $\phi_{\mathrm{true}} \coloneqq \mathsf{wp}_{\Xi,y}(\Gamma, \Delta \vdash e : \mathsf{B}, y = \texttt{true})$, and $\phi_{\mathrm{exit}} \coloneqq \mathsf{wp}^{\downarrow}_{\Xi,y}(\Gamma, \Delta \vdash e : \mathsf{B}, y = \texttt{false})$. See that $\neg\phi_{\mathrm{false}} \wedge \mathsf{W} \Rightarrow \mathsf{wp}^{\downarrow}_{\Xi,y}(c, \mathsf{W})$. Hence, we have the specification:

$$\Xi \triangleright \Gamma; \Delta \Vdash \left\{\neg\phi_{\mathrm{false}} \wedge \mathsf{W}\right\}\ c\ \ \downarrow\left\{\mathsf{W}\right\} \tag{5.1}$$

Now, consider the triple derived from the induction hypothesis:

$$\Xi, z_0 : \mathsf{Z} \triangleright \Gamma; \Delta \vdash \left\{\mathsf{wlp}_{(\Xi, z_0:\mathsf{Z}),\_}(c, \forall z : \mathsf{Z}\,.\,\psi \Rightarrow z < z_0)\right\}\ c\ \ \downarrow\left\{\forall z : \mathsf{Z}\,.\,\psi \Rightarrow z < z_0\right\}$$

If $\neg\phi_{\mathrm{false}} \wedge \mathsf{W} \wedge \psi[z_0/z]$ holds, by $\psi[z_0/z]$ and $\neg\phi_{\mathrm{false}}$, we get $\mathsf{wp}_{\Xi,\_}(c, \phi_{z_0})$. By the definition of ψ, since z_0 is the smallest index where ϕ_{z_0+1} holds, it holds that $\phi_{z_0} \Rightarrow (\forall z : \mathsf{Z}\,.\,\psi \Rightarrow z < z_0)$. By the

monotonicity, it holds that $\mathsf{wp}_{\Xi,\_}(c, \phi_{z_0}) \Rightarrow \mathsf{wp}_{\Xi,\_}(c, \forall z.\, \psi \Rightarrow z < z_0)$. Therefore, we have the implication $\neg\phi_{\text{false}} \wedge \mathsf{W} \wedge \psi[z_0/z] \Rightarrow \mathsf{wp}_{\Xi,\_}(c, \forall z : \mathsf{Z}\,.\, \psi \Rightarrow z < z_0)$. Therefore, by the rule of precondition weakening, we have the specification:

$$\Xi, z_0{:}\mathsf{Z} \triangleright \Gamma; \Delta \vdash \left\{\neg\phi_{\text{false}} \wedge \mathsf{W} \wedge \psi[z_0/z]\right\}\ c\ \ \downarrow\left\{\forall z : \mathsf{Z}\,.\, \psi \Rightarrow z < z_0\right\}$$

Together with the specification at 5.1, we derive the specification:

$$\Xi, z_0 : \mathsf{Z} \triangleright \Gamma; \Delta \vdash \left\{\neg\phi_{\text{false}} \wedge \mathsf{W} \wedge \psi[z_0/z]\right\}\ c\ \ \downarrow\left\{\mathsf{W} \wedge \forall z : \mathsf{Z}\,.\, \psi \Rightarrow z < z_0\right\}$$

See that the side-condition is direct from the definition of ψ. Hence, we have the desired specification. □

Chapter 6. Clerical in $\mathbf{Asm}(\mathbb{N}^{\mathbb{N}})$

Similar to Chapter 4, in this chapter, we devise an interpretation of Clerical in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$. Since the motivation and the general picture is already explained in Chapter 4, let us directly dive into the business.

To each data type τ, we define

$$[\![\mathsf{U}]\!] \coloneqq \mathbf{1} \qquad [\![\mathsf{B}]\!] \coloneqq \mathbf{2} \qquad [\![\mathsf{Z}]\!] \coloneqq \mathbf{Z} \qquad [\![\mathsf{R}]\!] \coloneqq \mathbf{R}$$

where $\mathbf{R}$ is any effective represented set of real numbers.

To each typing context Γ, we assume there is an assembly $[\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$ of $[\![\Gamma]\!]$ with the following computable morphisms:

1. $\mathsf{assign}_x : [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \times [\![\Gamma(x)]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \to [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$ such that $\mathsf{assign}_x(\gamma, v) = \gamma[x_i \mapsto x]$,
2. $\mathsf{extend}_{x:\tau} : [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \times [\![\tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \to [\![\Gamma, x : \tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$ such that $\mathsf{extend}_{x:\tau}(\gamma, v) = (\gamma, (x \mapsto v))$,
3. $\mathsf{value}_x : [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \to [\![\Gamma(\tau)]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$ such that $\mathsf{value}_x(\gamma) = \gamma(x)$,
4. $\mathsf{remove}_x : [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \to [\![\Gamma \restriction_{\mathrm{dom}(\Gamma)\setminus\{x\}}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$ such that $\mathsf{remove}_x(\gamma, x \mapsto w) = \gamma$,

Moreover, we assume the empty state () of $[\![\cdot]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$ is computable.

To each typing context $x_1 : \tau_1, \cdots, x_n : \tau_n$, there is a morphism $\mathsf{state} : [\![\tau_1]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \times \cdots \times [\![\tau_n]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \to [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$ such that $\mathsf{state}(v_1, \cdots, v_n) = (x_1 \mapsto v_1, \cdots, x_n \mapsto v_n)$. It can be done by repeatedly calling extend on the empty state. And, for any disjoint typing contexts Γ, Δ, there is a morhpism $\mathsf{join} : [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \times [\![\Delta]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \to [\![\Gamma, \Delta]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$ such that $\mathsf{join}(\gamma, \delta) = (\gamma, \delta)$. It can be done by repeatedly extending δ on γ.

6.1 Modified Powerdomain in $\mathbf{Asm}(\mathbb{N}^{\mathbb{N}})$

For a set S, we want to define an endofunctor $\mathsf{P}_\star(\square) : \mathsf{Asm}(\mathbb{N}^{\mathbb{N}}) \to \mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$ where for any assembly $\mathbf{A}$, the underlying set of $\mathsf{P}_\star(\mathbf{A})$ is $\mathbb{P}_\star(|\mathbf{A}|)$.

For any assembly $\mathbf{A}$, define $\mathsf{P}_\star(\mathbf{A})$ be the assembly induced from the injective function

$$\begin{array}{rcccl} \iota_{\mathbf{A}} & : & \mathbb{P}_\star(|\mathbf{A}|) & \to & |\natural\mathsf{M}(\flat\mathbf{A})| \\ & : & S & \mapsto & \begin{cases} \natural & \text{if } S = \mathfrak{e}, \\ S \setminus \{\bot\} \cup \{\flat\} & \text{if } \bot \in S, \\ S & \text{otherwise.} \end{cases} \end{array}$$

See that ι translate $\mathfrak{e}$ to $\natural$ and $\bot$ to $\flat$. By definition, the injective mapping appears as a morphism

$$\iota_{\mathbf{A}} : \mathsf{P}_\star(\mathbf{A}) \to \natural\mathsf{M}(\flat\mathbf{A})$$

Also, the rectifying operation, which is a retraction of $\iota_{\mathbf{A}}$, is computable:

$$\begin{array}{rcccl} r_{\mathbf{A}} & : & \natural\mathsf{M}(\flat\mathbf{A}) & \to & \mathsf{P}_\star(\mathbf{A}) \\ & : & S & \mapsto & \begin{cases} \mathfrak{e} & \text{if } S = \natural, \\ S \setminus \{\flat\} \cup \{\bot\} & \text{if } \flat \in S, \\ S \cup \{\bot\} & \text{if } S \text{ infinite}, \\ S & \text{otherwise.} \end{cases} \end{array}$$

By defining the action on morphisms

$$\mathsf{P}_\star(f : \mathbf{A} \to \mathbf{B}) \coloneqq r_{\mathbf{B}} \circ (\natural\mathsf{M}(\flat(f))) \circ \iota_{\mathbf{A}},$$

the mapping $\mathsf{P}_\star(\square)$ is an endofunctor in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$. Note that its definition is $\mathbb{P}_\star(f)$.

Also, see that the unit and the multiplication of $\mathbb{P}_\star()$ are computable. Hence, the triple $(\mathsf{P}_\star(\square), \eta, \mu)$ is a monad that their definitions form a monad in Set.

Moreover, since

$$\begin{array}{rcccl} \zeta_{\mathbf{A},\mathbf{B}} & : & (\mathbf{A} \to \mathbf{B}) & \to & (\mathsf{P}_\star(\mathbf{A}) \to \mathsf{P}_\star(\mathbf{B})) \\ & : & f & \mapsto & S \mapsto \bigcup_{x \in S} \begin{cases} \mathfrak{e} & \text{if } x = \mathsf{e}, \\ \{\bot\} & \text{if } x = \bot, \\ \{f(x)\} & \text{otherwise,} \end{cases} \end{array}$$

is computable, (by composing the extensions of $\natural, \mathsf{M}, \flat$), we can confirm that the endofunctor is a strong moand with the same α and β that of $\mathbb{P}_\star(\square)$. That means the definitions of the liftings are the liftings w.r.t. $\mathbb{P}_\star(\square)$.

Note, however, that $\mathsf{P}_\star(\square)$ fails to be a countably applicative functor.

For a morphism $f : \mathbf{A} \to \flat\mathbf{B}$, define $f^\ddagger : \mathbf{A} \to \mathsf{P}_\star(\mathbf{B})$ by

$$f^\ddagger(x) = \begin{cases} \{\bot\} & \text{if } x = \flat. \\ \{f(x)\} & \text{otherwise,} \end{cases}$$

which is computable by the realizer of $\eta^\natural \circ \eta^{\mathsf{M}} \circ f$.

For the domain-theoretic properties, see that the chain completeness is effective.

Lemma 6.1. To each assembly $\mathbf{A}$, there is a computable function $\mathsf{LIM}_{\mathbf{A}} : (\mathbf{N} \to \mathsf{P}_\star(\mathbf{A})) \to \mathsf{P}_\star(\mathbf{A})$ such that it on chains returns the limit of the chain. For a non-chain input, it returns $\mathfrak{e}$.

Proof. Suppose $\langle(\phi_i)_{i\in\mathbb{N}}\rangle$ is given where $\phi_i \Vdash_{\mathsf{P}_\star(\mathbf{A})} S_i$ and $(S_i)_{i\in\mathbb{N}}$ is a chain. Until we find an index n iterating from 0 such that $\langle(\phi_i)_{i\in\mathbb{N}}\rangle(n) \neq 0, 1$, we append 1 in the output tape. If there is m, k such that $\langle(\phi_i)_{i\in\mathbb{N}}\rangle(\langle m, k\rangle) \neq 0, 1$, which means $\phi_m(k) \neq 0, 1$, we stop the iteration and just append $\phi_m(k), \phi_m(k+1), \cdots$ in the output tape.

Now we need to argue that the above computation realizes $\mathsf{LIM}_{\mathbf{A}}$. Suppose $\langle(\phi_i)_{i\in\mathbb{N}}\rangle$ was a name of an increasing chain. If there is no index n such that $\langle(\phi_i)_{i\in\mathbb{N}}\rangle(n) \neq 0, 1$ holds, the computation prints $1^{\mathbb{N}}$ which is a name of $\mathfrak{e}$ and any set containing $\bot$. Also, if ϕ_i consists of only 0 or 1, it can represent only $\mathfrak{e}$ or a set with $\bot$. If $S_i = \mathfrak{e}$ for some i, then the limit of the sequence is $\mathfrak{e}$. Hence, the computation was correct. If $S_i \neq \mathfrak{e}$ for all i, then $\bot \in S_i$ for all i. Hence, the limit of the sequence contains $\bot$ which is represented by $1^{\mathbb{N}}$. Hence, the computation was correct in this case as well.

When there is an index $\langle m,k\rangle$ such that $\langle(\phi_i)_{i\in\mathbb{N}}\rangle m,k\rangle \neq 0,1$ holds, (assume $\langle m,k\rangle$ is the smallest such number), the computation produces $1^{\langle m,k\rangle} :: \phi_m(k),\phi_m(k+1),\cdots$. Since $\phi_m(k)\neq 0,1$, it is either $S_m = \mathfrak{e}$ or $\bot \neq x \in S$ and $\phi_m^{<<}$ is a name of $x \in \mathbf{A}$. If $S_m = \mathfrak{e}$, then the limit is $\mathfrak{e}$. Since any sequence represents $\mathfrak{e}$, the computation is correct. If $\bot \neq x \in S$ and $\phi_m^{<<}$ is a name of $x \in \mathbf{A}$, it is either that the limit contains x or the limit is $\mathfrak{e}$. Since $1^{\langle m,k\rangle} :: \phi_m(k),\phi_m(k+1),\cdots$ is a name of $\mathfrak{e}$ or any set containing x in $\mathsf{P}_\star(\mathbf{A})$, the computation is correct.

If the input was a name of a non-converging sequence, since the computation still produces an infinite sequence, which is a name of $\mathfrak{e}$, the computation is correct. □

Due to the fixed-point theorem, any morphism that is domain-theoretic-continuous $f : \mathsf{P}_\star(\mathbf{A}) \to \mathsf{P}_\star(\mathbf{A})$ has the least fixed-point in $\mathsf{P}_\star(\mathbf{A})$.

Lemma 6.2. For any assembly $\mathbf{A}$ and a continuously realizable function $f : |\mathsf{P}_\star(\mathbf{A})| \to |\mathsf{P}_\star(\mathbf{A})|$, if f is domain-theoretic-continuous, the least fixed-point is uniformly computable. I.e., there is a computable function $\mathsf{LFP}_{\mathbf{A}} : (\mathsf{P}_\star(\mathbf{A}) \to \mathsf{P}_\star(\mathbf{A})) \to \mathsf{P}_\star(\mathbf{A})$ that returns the least-fixed points of the domain-theoretic-continuous functions. For functions that are not domain-theoretic-continuous, it returns $\mathfrak{e}$.

Proof. By the least fixed-point theorem, the least-fixed point is the limit of the chain $(f^n(\{\bot\}))_{n\in\mathbb{N}}$. Hence, we can simply return $\mathsf{LIM}_{\mathbf{A}}(\lambda(n:\mathbf{N})f^n(\{\bot\}))$ where the repeated application can be done by a primitive recursion. □

Lemma 6.3. For any assembly $\mathbf{A}$ and a continuously realizable function $f : (\mathbf{A} \to \mathsf{P}_\star(\mathbf{A})) \to (\mathbf{A} \to \mathsf{P}_\star(\mathbf{A}))$, if f is point-wise domain-theoretic-continuous, the least-fixed point of f is uniformly computable. I.e., there is $\mathsf{LFP}_{\mathbf{A}\to\mathsf{P}_\star(\mathbf{A})} : \big((\mathbf{A} \to \mathsf{P}_\star(\mathbf{A})) \to \mathbf{A} \to \mathsf{P}_\star(\mathbf{A})\big) \to (\mathbf{A} \to \mathsf{P}_\star(\mathbf{A}))$ such that $\mathsf{LFP}_{\mathbf{A}\to\mathsf{P}_\star(\mathbf{A})}(f)$ is the least fixed-point of f if f is continuous w.r.t. the point-wise ordering. Otherwise,$\mathsf{LFP}_{\mathbf{A}\to\mathsf{P}_\star(\mathbf{A})}(f)(x) = \mathfrak{e}$ for any x.

Proof. For a domain-theoretic continuous function $f : (\mathbf{A} \to \mathsf{F}(\mathbf{A})) \to (\mathbf{A} \to \mathsf{F}(\mathbf{A}))$, by the fixed point theorem, the least-fixed point of it is the limit of $\lambda n.\ f^n(x \mapsto \{\bot\})$. Hence, we can define

$$\mathsf{LFP}_{\mathbf{A}\to\mathsf{P}_\star(\mathbf{A})}(f)(x) = \mathsf{LIM}(\lambda n : \mathbf{N}.\ (f^n(\lambda y.\ \{\bot\})))\ x\ .$$

□

6.2 Computability of Clerical

To each well-typed read-only expression $\Gamma \vdash e : \tau$, we interpret it as a morhpsim $[\![\Gamma \vdash e : \tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$ from $[\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$ to $\mathsf{P}_\star([\![\tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})})$ such that

$$\mathbf{\Gamma}([\![\Gamma \vdash e : \tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}) = [\![\Gamma \vdash e : \tau]\!]$$

holds in Set. And, to each well-typed read-write expression $\Gamma;\Delta \Vdash c : \tau$, we interpret it as a morhpsim $[\![\Gamma;\Delta \vdash c : \tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$ from $[\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}$ to $[\![\Delta]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \to \mathsf{P}_\star([\![\tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})})$ such that

$$\mathbf{\Gamma}([\![\Gamma;\Delta \Vdash c : \tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}) = [\![\Gamma;\Delta \Vdash c : \tau]\!]$$

holds in Set.

First, when the well-typedness $\Gamma \vdash e : \tau$ is from $\Gamma;\cdot \Vdash e : \tau$, we do

$$[\![\Gamma \vdash e : \tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} = \lambda\gamma.\ \pi_1^\dagger \circ [\![\Gamma;\cdot \Vdash e : \tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}\gamma\cdot$$

And, when the well-typedness $\Gamma; \Delta \Vdash c : \tau$ is from $\Gamma, \Delta \vdash c : \tau$, we define

$$[\![\Gamma; \Delta \vdash c : \tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} = \lambda\gamma.\, \lambda\delta.\, \left(\delta, {}^{\dagger_2}\, [\![\Gamma; \cdot \Vdash e : \tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}(\gamma, \delta)\right)$$

See that the atomic operations are clear:

$$\begin{aligned}
[\![\Gamma \vdash \mathtt{true} : \mathsf{B}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &= (\lambda\gamma.\, tt)^{\dagger} : [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \to \mathsf{P}_{\star}(\mathbf{2}) \\
[\![\Gamma \vdash \mathtt{true} : \mathsf{B}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &= (\lambda\gamma.\, ff)^{\dagger} : [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \to \mathsf{P}_{\star}(\mathbf{2}) \\
[\![\Gamma \vdash k : \mathsf{Z}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &= (\lambda\gamma.\, k)^{\dagger} : [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \to \mathsf{P}_{\star}(\mathbf{Z}) \\
[\![\Gamma \vdash \iota(e) : \mathsf{R}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &= (\lambda\gamma.\, \iota^{\dagger}[\![\Gamma \vdash t : \mathsf{Z}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}\gamma) : [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \to \mathsf{P}_{\star}(\mathbf{R}) \\
[\![\Gamma \vdash e_1 \odot e_2 : \mathsf{Z}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &= (\lambda\gamma.\, [\![\Gamma \vdash e_1 : \mathsf{Z}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}\gamma \odot^{\dagger} [\![\Gamma \vdash e_2 : \mathsf{Z}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}\gamma) : [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \to \mathsf{P}_{\star}(\mathsf{Z}) \\
[\![\Gamma \vdash e_1 < e_2 : \mathsf{B}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &= (\lambda\gamma.\, [\![\Gamma \vdash e_1 : \mathsf{Z}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}\gamma <^{\dagger} [\![\Gamma \vdash e_2 : \mathsf{Z}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}\gamma) : [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \to \mathsf{P}_{\star}(\mathsf{Z}) \\
[\![\Gamma \vdash e_1 = e_2 : \mathsf{Z}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &= (\lambda\gamma.\, [\![\Gamma \vdash e_1 : \mathsf{Z}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}\gamma =^{\dagger} [\![\Gamma \vdash e_2 : \mathsf{Z}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}\gamma) : [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \to \mathsf{P}_{\star}(\mathsf{Z}) \\
[\![\Gamma \vdash e_1 \boxdot e_2 : \mathsf{R}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &= (\lambda\gamma.\, [\![\Gamma \vdash e_1 : \mathsf{R}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}\gamma \boxdot^{\dagger} [\![\Gamma \vdash e_2 : \mathsf{R}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}\gamma) : [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \to \mathsf{P}_{\star}(\mathsf{R}) \\
[\![\Gamma \vdash e^{-1} : \mathsf{R}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &= (\lambda\gamma.\, ((([\![\Gamma \vdash e : \mathsf{R}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}\gamma)^{-1}\downarrow_{\flat}^{\sharp\dagger}) : [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \to \mathsf{P}_{\star}(\mathsf{R}) \\
[\![\Gamma \vdash e_1 \hat{<} e_2 : \mathsf{B}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} &= (\lambda\gamma.\, [\![\Gamma \vdash e_1 : \mathsf{R}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}\gamma < \downarrow_{\flat}^{\sharp\dagger} [\![\Gamma \vdash e_2 : \mathsf{R}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}\gamma) : [\![\Gamma]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} \to \mathsf{P}_{\star}(\mathsf{B})
\end{aligned}$$

And, for a variable x, we have

$$[\![\Gamma \vdash x : \tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} = \lambda\gamma.\, \mathsf{value}_x^{\dagger}(\gamma)$$

Interpreting the limit operator is a little complicated that in the category of sets, $\mathbb{P}_{\star}(\square)$ was countably applicative but $\mathsf{P}_{\star}(\square)$ is not in $\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})$. Hence, we cannot use the same construction. Recall that $\natural\mathsf{M}\natural$ is countably applicative where we can lift the $\natural$ extended lim to

$$(\mathsf{lim}\downarrow_{\natural})^{\dagger} : (\mathbf{N} \to \natural\mathsf{M}(\natural\mathbf{R})) \to \natural\mathsf{M}(\natural\mathbf{R})\ .$$

For a mapping $g : \mathbf{A} \to \mathbf{B}$, define $g^{\omega} \coloneqq (\lambda(f : \mathbf{N} \to \mathbf{A}).\ \lambda(n : \mathbf{N}).\ g(f\ n)) : (\mathbf{N} \to \mathbf{A}) \to (\mathbf{N} \to \mathbf{B})$. Precomposing $(\natural\mathsf{M}(\kappa^{\flat,\natural}))^{\omega}$ yields

$$(\mathsf{lim}\downarrow_{\natural})^{\dagger} \circ \natural\mathsf{M}(\kappa^{\flat,\natural})^{\omega} : (\mathbf{N} \to \natural\mathsf{M}(\flat\mathbf{R})) \to \natural\mathsf{M}(\natural\mathbf{R}).$$

And, define an auxiliary mapping:

$$\begin{array}{rcccl}
J & : & \natural\mathsf{M}(\natural\mathbf{R}) & \to & \natural\mathsf{M}(\flat\mathbf{R}) \\
 & : & S & \mapsto & \begin{cases} \natural & \text{if } S = \natural \vee \natural \in S, \\ S & \text{otherwise.} \end{cases}
\end{array}$$

which identifies $\natural$ and $\flat$ in S. Then, we have

$$r_{\mathbf{R}} \circ J \circ (\kappa^{\sharp,\natural} \circ \mathsf{lim})^{\dagger} \circ \natural\mathsf{M}(\kappa^{\flat,\natural})^{\omega} \circ (s_{\mathbf{R}})^{\omega} : (\mathbf{N} \to \mathsf{P}_{\star}(\mathbf{R})) \to \mathsf{P}_{\star}(\mathbf{R})$$

Check that its definition coincides with the semantics used in Section 5.3.3. Let us denote $\iota_{\mathbf{N},\mathbf{Z}} : \mathbf{N} \to \mathbf{Z}$ for the subset inclusion. Then, we can define

$$\begin{aligned}
&[\![\Gamma \vdash \mathtt{lim}\ x\,.\,e : \mathsf{R}]\!] = \\
&\quad \lambda\gamma.\, r_{\mathbf{R}} \circ J \circ (\kappa^{\sharp,\natural} \circ \mathsf{lim})^{\dagger} \circ \natural\mathsf{M}(\kappa^{\flat,\natural})^{\omega} \circ (s_{\mathbf{R}})^{\omega}(\lambda(n : \mathbf{N}).\, [\![\Gamma, x{:}\mathsf{Z} \vdash e : \mathsf{R}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}(\mathsf{extend}_x(\gamma, \iota_{\mathbf{N},\mathbf{Z}}(n))))
\end{aligned}$$

For the guarded nondeterminism, let us define an extension of Cond function

$$\begin{array}{rcccl} \mathsf{Cond}^n_S & : & \mathbb{N} \times S^n & \to & \natural S \\ & : & (k, x_1, \cdots, x_n) & \mapsto & \begin{cases} x_1 & \text{if } n = 1, \\ x_2 & \text{if } n = 2, \\ \vdots & \vdots & \\ x_n & \text{if } n = k. \\ \natural & \text{otherwise} \end{cases} \end{array}$$

which is trivially computable. Define,

$$[\![\texttt{case}\ e_1 \Rightarrow c_1 \mid \cdots \mid e_n \Rightarrow c_n\ \texttt{end}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} =$$

$$\lambda\gamma.\ \lambda\delta.$$

$$\left(\mathsf{Cond}^n_{\mathsf{P}_\star(S)}\right)^{\dagger_1}((\mathsf{choice}\,\natural_\flat)^\dagger([\![e_1]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}(\gamma,\delta), \cdots, [\![e_n]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}(\gamma,\delta)), [\![c_1]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}\gamma\,\delta, \cdots, [\![c_n]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}\gamma\,\delta)$$

For any assembly $\mathbf{A}$, define $j_{\mathbf{A}} : \mathbf{A} \times \mathbf{1} \to \mathbf{A}$ for the projection map and $k_{\mathbf{A}} : \mathbf{A} \to \mathbf{A} \times \mathbf{1}$ for the map $x \mapsto (x, *)$. Of course, they are computable isomorphisms. For any $b : \mathbf{A} \to \mathsf{P}_\star(\mathbf{2})$ and $c : \mathbf{A} \to \mathsf{P}_\star(\mathbf{A} \times \mathbf{1})$, define

$$\mathbf{W}(b, c) : \lambda(f : \mathbf{A} \to \mathsf{P}_\star(\mathbf{A} \times \mathbf{1}))\text{.}\ \mathsf{Cond}^{\dagger_1}_{\mathsf{P}_\star(\mathbf{A}\times\mathbf{1})} \circ (b \times (f^\dagger \circ j^\dagger \circ c) \times k^\dagger)$$

From Lemma 5.7, we know that $\mathbf{W}(b, c) : (\mathbf{A} \to \mathsf{P}_\star(\mathbf{A} \times \mathbf{1})) \to (\mathbf{A} \to \mathsf{P}_\star(\mathbf{A} \times \mathbf{1}))$ is a domain-theoretic continuous function. Hence, $\mathsf{LFP}_{\mathbf{A}\to\mathsf{P}_\star(\mathbf{A}\times\mathbf{1})}(\mathbf{W}(b, c))$ is the least fixed-point of the mapping. Hence, we can define

$$[\![\Gamma; \Delta \Vdash \texttt{while}\ e\ \texttt{do}\ c\ \texttt{end}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} = \lambda\gamma.\ \mathsf{LFP}_{[\![\Delta]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}\to\mathsf{P}_\star([\![\Delta]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}\times\mathbf{1})}(\mathbf{W}(\lambda\delta.\ [\![e]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}(\gamma,\delta), [\![c]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}\gamma))$$

using Lemma 6.3

For an assignment, $x := e$, we can simply let

$$[\![\Gamma; \Delta \Vdash x := e : \mathsf{U}]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} = \lambda\gamma.\ \lambda\delta.\ \left(\mathsf{assign}^{\dagger_2}_x(\delta, [\![\Gamma, \Delta \vdash e : \tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}(\gamma,\delta)), ^{\dagger_1} *\right)$$

And, for a local variable creation,

$$\begin{array}{l} [\![\Gamma; \Delta \Vdash \texttt{new var}\ x := e\ \texttt{in}\ c : \tau]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})} = \\ \quad \lambda\gamma.\ \lambda\delta.\ (\mathsf{remove}_x \times \mathrm{id})^\dagger \circ \left([\![c]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}{}^{\dagger_2}\gamma\,(\mathsf{extend}^{\dagger^2}_x(\delta, [\![e]\!]_{\mathsf{Asm}(\mathbb{N}^{\mathbb{N}})}(\gamma,\delta)))\right) \end{array}$$

See that their definitions are the denotations of the expressions.

Chapter 7. Reliable Symmetric Matrix Eigenproblem

7.1 Introduction

The computational problem of matrix diagonalization plays important roles in many areas of science and engineering: from quantum physics, artificial intelligence in computer science, to random matrix theory in pure mathematics. The problem consists of two natural subproblems: computing each eigenvalue with its multiplicity and computing the associated eigenspace of each eigenvalue. We are interested in the task of solving the eigenproblem *rigorously*. That is, on all (even degenerate) inputs, within time bounded by a guaranteed number of (bit) operations, produce output satisfying any given error bound. Note that the general symmetric eigenspace problem is discontinuous/unstable already in the 2×2 case:

$$A(\epsilon) := \exp(-1/\epsilon^2) \cdot \begin{pmatrix} \cos(2/\epsilon) & \sin(2/\epsilon) \\ \sin(2/\epsilon) & -\cos(2/\epsilon) \end{pmatrix}, \quad A(0) := \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

The problem is known to become continuous when restricted to symmetric $d \times d$ matrices having, among its d eigenvalues including multiplicities, precisely k pairwise distinct ones [ZB04, §3.5] for any fixed $k \in \mathbb{N}$. We establish:

Theorem 7.1. There exists an algorithm that, given any symmetric $d \times d$ interval matrix A of componentwise widths $2^{-m(A,p)}$ having exactly k of its d eigenvalues distinct, outputs k pairwise disjoint intervals of widths 2^{-p} containing said eigenvalues; and outputs d interval vectors of componentwise widths 2^{-p} forming an eigenbasis, where $m(A,p) \in \mathcal{O}\big(d^2(p + d^2 + d \log 1/\Delta(A) + |\log \|A\|_F|)^2\big)$. In this case the bit-cost of our algorithm is bounded by $C(A,n) \in \mathcal{O}\big(d^2(\sqrt{n}/d + d^2 + \log \|A\|_F)\mathtt{M}(d + n + \log \|A\|_F)\big)$. □

7.1.1 QR Algorithm and Wilkinson Shift

A symmetric matrix can be efficiently reduced to a similar symmetric tridiagonal matrix using sequential Householder reflections. (Here, the similarity is a technical term; two matrices are similar if they share identical eigenvalues counting the multiplicities.) For a tridiagonal matrix T, consider its QR decomposition $Q \cdot R := T$. The matrix $R \cdot Q$ is similar to T. The procedure of repeatedly performing this similarity transformation is the well-known QR algorithm [Wat82]. We call a single step in the iterative procedure, which is to compute T' from T, to be a QR step. When the algorithm makes the off-diagonal entries small enough, Weyl's inequality is used to obtain rigorous approximations to the eigenvalues. However, it is *not a total algorithm* in that there are matrices which make it never converges.

To make the QR algorithm *total* and to speed up the rate of convergence, from linear to quadratic or cubic, shifting is often used [GVL96, p. 456]. A shift λ is a real-valued function on the set of matrices. Given a shift, the QR algorithm with the shift is defined as follows. From the usual QR algorithm, at the beginning of each QR step, when T is the matrix to be processed, perform the QR decomposition on the shifted matrix $Q \cdot R := T - \lambda(T)$ instead. Then, compute $T' := R \cdot Q + \lambda(T)$. Again, $T \mapsto T'$ is a similarity transformation.

In 1968, Wilkinson showed that when *Wilkinson shift* is used, the QR algorithm with the shift guarantees a quadratic rate of convergence for *any* symmetric tridiagonal matrices [Wil68]. The Wilkinson

shift λ_{Wilk} is defined as follows:

$$\lambda_{\text{Wilk}}(T) := \text{the eigenvalue of } T_{d-1:d,d-1:d} \text{ which is closer to } T_{d,d} \tag{7.1}$$

Here, T is a $d \times d$ dimensional matrix, $T_{d-1:d,d-1:d}$ is the 2×2 bottom-right submatrix of T and $T_{d,d}$ is (d, d) entry of T. When the two eigenvalues are equally distanced from the last entry, any of the two can be selected.

7.1.2 Reliable Computation using Intervals

A real number is an infinite object that cannot be represented exactly by using any finite representation. Hence, one must finitely *approximate* it to represent and compute with on a digital computer. A common practice of approximating it with a finite precision floating point number often fails to be reliable due to the inherent rounding errors in its computation [Rum88, LW02]. A tedious rounding error analysis on a floating-point computation enables us to catch the magnitude of the rounding errors that occur during the computation. Nonetheless, it is unavoidable to face a total erroneous result for input sensitive cases.

Instead, interval computation can replace floating-point computation for the purpose of reliability [MKC09, Moo14]; a finite approximation of a real number is an interval with dyadic endpoints that contains the real number. Computing a real number is realized by computing an interval that is promised to contain the real number. Hence, we get a rigorous bound on the real number. By forcing the endpoints to be dyadic rational numbers, interval computation reduces to integer computation such that it can be simulated on a digital computer exactly and its realistic run-time can be obtained by counting the number of bit operations.

A side-effect of carrying out real number computation using interval computation is partiality. We define the order of two intervals only when they are disjoint; seeing the two intervals as some finite approximations of real numbers, when they are disjoint, we can, for sure, decide which of the real numbers are greater. However, when the intervals overlap, we cannot decide which among the real numbers are approximated by the intervals is greater. Hence, when we are comparing identical real numbers, no matter how tight the intervals approximating the real number are, it always fails.

Let $\mathbb{ID} := \{[a/2^n, b/2^m] : a, b, n, m \in \mathbb{Z}\}$ be the set of dyadic intervals. For vectors of intervals $\tilde{x}, \tilde{y} \in \mathbb{ID}^d$, the membership and inclusion relations are defined entry-wise: $x \in \tilde{x} :\Leftrightarrow x_i \in \tilde{x}_i$ for all i, and $\tilde{x} \subseteq \tilde{y} :\Leftrightarrow \tilde{x}_i \subseteq \tilde{y}_i$ for all i. For a set S, let us write $S_\perp$ to denote $S \cup \{\perp\}$. Let π_j be the canonical projection function on vectors that returns the jth entry. For the set $\mathbb{ID}^d_\perp$, we extend the inclusion relation to be $\tilde{x} \subseteq \perp$ for any $\tilde{x} \in \mathbb{ID}^d_\perp$. Furthermore, for an interval $\tilde{x} \coloneqq [a, b] \in \mathbb{ID}$, we write $w(\tilde{x}) := b - a$ be the width of $\tilde{x}$. We extend it to $\tilde{x} \in \mathbb{ID}^d$ by defining $w(\tilde{x}) := \max\{w(\tilde{x}_i) : i \in [1, d]\}$. The relations and the function w are again extended to matrices by treating a matrix as vector with any, but a fixed, index traversal.

A function $\tilde{f} : \mathbb{ID}^{d_r} \times \mathbb{Z}^{d_z} \to \mathbb{ID}^{e_r} \times \mathbb{Z}^{e_z}_\perp$ models an interval computation which receives d_r intervals and d_z integers, and returns e_r intervals and e_z integers. For any $(\tilde{x}, y) \in \mathbb{ID}^{d_r} \times \mathbb{Z}^{d_z}$, the computation on them yields $\tilde{f}(\tilde{x}, y)$ where the case $\tilde{f}(\tilde{x}, y) = \perp$ represents the computation failing. An interval function $\tilde{f} : \mathbb{ID}^{d_r} \times \mathbb{Z}^{d_z} \to \mathbb{ID}^{e_r} \times \mathbb{Z}^{e_z}_\perp$ *realizes* a real function $f : \mathbb{R}^{d_r} \times \mathbb{Z}^{d_z} \to \mathbb{R}^{e_r} \times \mathbb{Z}^{e_z}$ if (i) it is sound (as an approximate computation): for any $x \in \tilde{x}$, $\pi_j(f(x, y)) \in \pi_j(\tilde{f}(\tilde{x}, y))$ holds for all $1 \leq j \leq d_r$ and $\pi_j(f(x, y)) = \pi_j(\tilde{f}(\tilde{x}, y))$ holds for all $d_r < j \leq d_r + d_z$; (ii) it is inclusion-monotone: if $f(\tilde{y}, z) \neq \perp$, then $\pi_j(f(\tilde{x}, z)) \subseteq \pi_j(f(\tilde{y}, z))$ for any $1 \leq j \leq d_r$, $\tilde{x} \subseteq \tilde{y}$ and z; and (iii) it is (chain-)continuous: for each x, z and $p \in \mathbb{N}$, there is $m \in \mathbb{N}$ such that for any $\tilde{x} \ni x$ such that $w(\tilde{x}) < 2^{-m}$, it holds that $w(\tilde{f}(\tilde{x}, z)) < 2^{-p}$.

In other words, there is a monotone function $m : \mathbb{R}^{d_r} \times \mathbb{Z}^{d_z} \times \mathbb{N} \to \mathbb{N}$ where for each x, z and $p \in \mathbb{N}$, it holds that for any $\tilde{x} \ni x$ where $w(\tilde{x}) < 2^{-m(x,z,p)}$, it holds that $w(\tilde{f}(\tilde{x}, z)) < 2^{-p}$. We call the function m to be a local *modulus of continuity* of $\tilde{f}$.

The intuition behind the notion of realizing is as follows. The function f is what we want to compute in the ideal world. Input real numbers $x \in \mathbb{R}^d$ will get approximated by some dyadic intervals $\tilde{x} \in \mathbb{ID}^d$ where $x \in \tilde{x}$. Then, we run the computation for $\tilde{f}$ and get intervals $\tilde{f}(\tilde{x})$ when the computation succeeds. Due to (i), we guarantee that the result of the computation $\tilde{f}(\tilde{x})$ contains $f(x)$. Due to (ii), the approximation is consistent; namely, when we redo the computation on better approximations, the resulting approximations are also better. The continuity (iii) ensures that the computation is *total* in the sense that when x is approximated with good enough precision, when $w(\tilde{x})$ is small enough, the computation succeeds. Also, when we want arbitrarily good results, if we want the output intervals' width to be less than 2^{-p} for any natural number p, we can proceed with the computation with initial intervals' widths less than $2^{-m(x,p)}$ where m is a modulus of continuity for the interval function.

Given an *algorithm* over real numbers, the trivial conversion of it to an interval algorithm via changing reals to intervals and real arithmetic to interval arithmetic achieves (i) and (ii) automatically but not (iii). (See Section 7.3.2.) In our context, when the real matrix we want to diagonalize has its submatrix's eigenvalues equally distanced from its last entry, then whichever precision we proceed, *the interval computation for obtaining the Wilkinson shift fails.*

7.1.3 Related Works and Our Contributions

In order to overcome this infeasibility of Wilkinson shift in the context of reliable computing, we devise a new shifting which we call *fuzzy Wilkinson shift*. (See Section 7.4.) It is fuzzy in the sense that when the two distances (in Equation 7.1) are not strictly comparable (when the two intervals representing the distances overlap) with some criteria, we let any of the two be the shift. We prove that the QR algorithm with this fuzzy shift also obtains a similar convergence rate.

In the end, we devise an interval algorithm realizing matrix diagonalization problem which uses the QR algorithm as its subprocedure. When an interval matrix is given as an approximation of a real symmetric matrix, it computes intervals that rigorously contain real eigenvalues and orthogonal bases for eigenspaces. A modulus of continuity is obtained, which says how small the widths of the intervals in the input interval matrix should be in order to guarantee the output intervals width bounded by 2^{-p} for any natural number p. And, the computation's realistic run-time is obtained by counting the number of bit operations. It answers the question of how precise does an input real matrix should be approximated and how many bit operations are needed to compute 2^{-p} approximations of its eigenvalues and eigenvectors.

The approach of seeing an interval as a finite approximation of a real number and a real function as an interval function with certain property can be found in *domain theory* [DG96, Eda97, Sco70] and *interval analysis* [Moo66]. It is closely related to *computable analysis* [Wei00] and *exact real number computation* [BCRO86, BC88] which studies computing over real number rigorously; e.g., an implementation [Mül00] uses repeated interval computation to achieve errorless computation over reals. We follow this approach in that we are interested in the case where the widths of input intervals can be arbitrarily small. Requiring the number of distinct eigenvalues as additional input can be found in and is justified in [Zie12, ZB04].

This perspective produces major differences from the existing works from interval analysis. We use interval computation to reliably realize real number computation, which only exists in an ideal world. For example, solving the interval eigenvalue problem from [Dei91] requires a subroutine for computing

the exact eigenvalue of some real matrix Interval Gaussian algorithms are studied in [CM06] and in many other works. However, without full pivot searching, it lacks the continuity (iii) property.

Before, the complexity of the eigenproblem has been obtained as a bound on the number of algebraic operations in an algebraic model [PC99]. The number of bit operations has been counted in [SS18]; however, there, the problem is restricted to matrices with algebraic real entries. Our result applies to any symmetric matrices, even with transcendental entries. Our work is based on integer computation, hence it can be implemented as it is and achieves reliable computation. Being a small variant of the QR algorithm, it can be practically used. By counting the number of bit operations, the obtained complexity bound is realistic.

7.2 Problem Statement and Overview

We decompose our problem into the two subproblems:

Definition 7.1.

1. Given a pair $(\tilde{A}, k) \in \mathbb{ID}^{d\times d} \times \mathbb{N}$ where the interval matrix $\tilde{A}$ contains a symmetric matrix A with exactly k distinct eigenvalues, compute k pairs $(\tilde{\lambda}_i, \mu_i)_{i=1,\cdots,k} \in (\mathbb{ID} \times \mathbb{N})^k$ such that the intervals are disjoint and each interval $\tilde{\lambda}_i$ contains an eigenvalue λ_i of A whose multiplicity is μ_i.

2. Given a pair $(\tilde{A}, k) \in \mathbb{ID}^{d\times d} \times \mathbb{N}$ where the interval matrix $\tilde{A}$ contains a *diagonalizable* matrix A whose rank is k, compute k interval vectors $\tilde{x}_1, \cdots, \tilde{x}_k$, such that $\vec{0} \notin \tilde{x}_i$ for all i and each $\tilde{x}_i$ contains a vector x_i such that $\{x_1, \cdots, x_k\}$ span the kernel of A.

In Section 7.3, we define dyadic interval computation with correct rounding that is used throughout this paper. In Section 7.4, the fuzzy Wilkinson shift is defined with a lemma stating that the shift is sound. In Section 7.5, we define and analyze Algorithm separate_eig for solving the first subproblem. In Section 7.6, we analyze an interval Gaussian algorithm interval_gaussian with fuzzy and full pivot searching, which is for solving the second subproblem.

Combining the two algorithms, we get the following interval algorithm:

Algorithm 1: interval_eig$(\tilde{A}, k)$

$(\tilde{\lambda}_1, \mu_1), \cdots, (\tilde{\lambda}_k, \mu_k) \coloneqq$ separate_eig$(\tilde{A}, k)$
for $i := 1 \to k$ **do** $(\tilde{x}_1, \cdots \tilde{x}_{\mu_i}) \coloneqq$ interval_gaussian$(\tilde{A} - \tilde{\lambda}_i, d - \mu_i)$
return $(\tilde{\lambda}_i, (\tilde{x}_{i_1}, \tilde{x}_{i_2}, \cdots, \tilde{x}_{i_{\mu_i}}))_{i=1,\cdots,k}$

To analyze the behaviour of the algorithm, we need to pick a natural parameter of the problem. It is promised that the ideal A in the input interval matrix has exactly k distinct eigenvalues. We devise a quantitative measure on how strong the property is:

Notation 7.1.

1. For a real symmetric matrix A, let $\Lambda(A)$ be the set of the eigenvalues of A and $\|A\|_F$ be the Frobenius norm of A. Let us write $\Delta(A)$ to denote the relative eigenvalue separation of A: $\Delta(A) := \min_{\lambda_1\neq\lambda_2}\{|\lambda_1 - \lambda_2| \ : \ \lambda_1, \lambda_2 \in \Lambda(A)\}/\|A\|_F$.

2. For a natural number ℓ, let us write $\mathtt{M}(\ell)$ be the number of bit operations for multiplying two ℓ bit integers. Recall that $\mathtt{M}(\ell)$ can be regarded as $\mathcal{O}(\ell \log \ell)$ [HVDH20].

3. For $n \in \mathbb{N}$, let $\mathbb{ID}_n := \{[a/2^n, b/2^n] \mid a, b \in \mathbb{Z}, a < b\}$ be the set of intervals whose endpoints are dyadic numbers of exponent n. Defining it to be closed under operations (in Section 7.3), n can be seen as a computation precision.

Theorem 7.2. Consider an interval matrix $\tilde{A} \in \mathbb{ID}_n^{d\times d}$ and a natural number k where $\tilde{A}$ contains a symmetric matrix A that admits exactly k distinct eigenvalues.

1. The interval algorithm separate_eig on $(\tilde{A}, k)$ for separating distinct eigenvalues admits the following modulus of continuity:

$$m_1(A,p) \in \mathcal{O}\big(d^2(p + \log 1/\Delta(A) + \log d + |\log \|A\|_F|)^2\big)$$

   I.e., the algorithm succeeds and returns intervals whose widths are bounded by 2^{-p} when the width of the interval matrix $\tilde{A}$ is less than $2^{-m_1(A,p)}$. Under this condition, the number of bit operations is bounded by C_1 where $C_1(A,n) \in \mathcal{O}\big(d^2(\sqrt{n}/d + d + \log \|A\|_F) \cdot \mathtt{M}(\log d + \log \|A\|_F + n)\big)$.

2. The interval algorithm interval_gaussian on $(\tilde{A} - \tilde{\lambda}, \mu)$ for computing a basis of the μ-dimensional eigenspace of A associated with $\lambda \in \tilde{\lambda}$ admits the following modulus of continuity:

$$m_2(A,p) \in p + \mathcal{O}(d^2 + d \log 1/\Delta(A) + |\log \|A\|_F|).$$

   I.e., the algorithm succeeds when the widths of the interval matrix $\tilde{A}$ and the interval $\tilde{\lambda}$ are less than $2^{-m_2(A,p)}$ and returns intervals whose widths are bounded by 2^{-p}. Under this condition, the number of bit operations is bounded by C_2 where $C_2(A,n) \in \mathcal{O}\big(d^3 \cdot \mathtt{M}(d + n + \log \|A\|_F)\big)$.

Note that the conditions on the widths of intervals are automatically imposed to n as well; the unit width 2^{-n} in $\mathbb{ID}_n$ should be smaller than the required widths of intervals.

Theorem 7.2-2 suggests that when the interval eigenvalues have their widths less than $2^{-m_2(A,p)}$, the eigenvectors will be computed with their widths bounded by 2^{-p}. And, Theorem 7.2-1 suggests that when the width of the input interval matrix's entries are less than $2^{-m_1(A,m_2(A,p))}$, the condition is satisfied. Considering the interval Gaussian algorithm should be applied for $k \leq d$ times, the overall bit-cost can be composed easily as Theorem 7.1.

Remark 7.1. The formulation of our problem follows from our motivation to solve the eigenproblem of an ideal matrix $A \in \mathbb{R}^{d\times d}$ when it is given to us by $\tilde{A} \in \mathbb{ID}_n^{d\times d}$ such that $A \in \tilde{A}$. It can be rearranged in such a way that k is the minimum number of distinct eigenvalues of symmetric matrices in $\tilde{A}$ and A is any symmetric matrix in $\tilde{A}$ that admits exactly k distinct eigenvalues, which is more similar to the setting of [Dei91].

7.3 Interval Computation and Fuzziness

7.3.1 Dyadic Intervals

We consider computations on intervals whose endpoints are dyadic numbers, which works as the standard interval computation as in [AH84] but with correct rounding. When $\tilde{x} := [a_1/2^n, a_2/2^n]$ and

$\tilde{y} := [b_1/2^n, b_2/2^n]$, we define the primitive operations as follows:

$$\begin{aligned}
\tilde{x} \;+\; \tilde{y} &:= [(a_1 + b_1)/2^n, (a_2 + b_2)/2^n] \\
\tilde{x} \;-\; \tilde{y} &:= [(a_1 - b_2)/2^n, (a_2 - b_1)/2^n] \\
\tilde{x} \;*\; \tilde{y} &:= [\lfloor \min_{i,j}(a_i b_j)/2^n \rfloor / 2^n, \lceil \max_{i,j}(a_i b_j)/2^n \rceil / 2^n] \\
\tilde{x} \;/\; \tilde{y} &:= \begin{cases} [\min_{i,j}(\lfloor a_i \cdot 2^n / b_j \rfloor)/2^n, \max_{i,j}(\lceil a_i \cdot 2^n / b_j \rceil)/2^n] & \text{if } 0 \notin [b_1, b_2] \\ \bot & \text{otherwise,} \end{cases} \\
\sqrt{\tilde{x}} &:= \begin{cases} \left[\left\lfloor \sqrt{a_1 \cdot 2^n} \right\rfloor / 2^n, \left\lceil \sqrt{a_2 \cdot 2^n} \right\rceil / 2^n\right] & \text{if } a_1 \geq 0 \\ \bot & \text{otherwise.} \end{cases}
\end{aligned}$$

The absolute value operator $|\tilde{x}|$ and the squaring operator $\tilde{x}^2$ are expected to make the resulting intervals' lower endpoints greater than or equal to 0. When it is not ambiguous, we often write $\tilde{x}\ \tilde{y}$ or$\tilde{x} \cdot \tilde{y}$ for $\tilde{x}\ *\ \tilde{y}$. Considering the partiality in the order comparisons, when we have an instruction of the form **if** $\tilde{x}\ >\ \tilde{y}$ **then** C_1 **else** C_2, we let C_1 execute only when $a_1 > b_1$. And, otherwise, we let C_2 execute.

Note that the operations, can be computed exactly using integer computations; we can count the number of bit operations of each interval operation; the number of bit operations of $\tilde{x}\ op\ \tilde{y}$ is bounded by $\mathcal{O}\big(C_{op}(\max(\log m(\tilde{x}), \log m(\tilde{y})) + n)\big)$ where $op \in \{+, -, *, /\}$ is an operation on integers or intervals, $m(\tilde{x}) := \max(|a_1|, |a_2|)/2^n$ is the magnitude of $\tilde{x}$, and $C_{op}(\ell)$ is the number of bit operations for performing op on ℓ bit integers. We take $C_+(\ell) = C_-(\ell) \leq C_*(\ell) = \mathtt{M}(\ell) \leq C_/(\ell) \in \mathcal{O}(\mathtt{M}(\ell) \log \mathtt{M}(\ell))$ for our analysis [BZ98].

7.3.2 Fuzzy Sign

Consider the function of computing the sign of a real number: $\mathrm{sign} : \mathbb{R} \ni x \mapsto 1$ if $x \geq 0$ and -1 if $x < 0$. There is no interval function $\mathrm{sign} : \mathbb{ID} \to \{0, 1\}$ that realizes the function. It is because when $x = 0$, any interval approximation of x contains 0. Hence, it cannot be distinguished whether x is greater than or less than 0.

One alternative approach is to let sign be fuzzy in that we give up to compute the sign of a real number exactly but let there be some tolerance factor such that when the real number is close to 0 w.r.t. the factor, we let the function returns any of 0 or 1 (nondeterministically). Namely, the fuzzy variant of the sign function [YSS13]:

$$\overline{\mathrm{sign}} : (x, \epsilon) \mapsto \begin{cases} 1 & \text{if } x > -\epsilon, \\ -1 & \text{if } x < \epsilon. \end{cases}$$

Observe that the above fuzzy sign is realizable by the interval function:

$$\underline{\mathrm{sign}} : (\tilde{x}, \tilde{\epsilon}) \mapsto \textbf{if } \tilde{x}\ >\ -\ \tilde{\epsilon} \textbf{ then } 1 \textbf{ else if } \tilde{x}\ <\ \tilde{\epsilon} \textbf{ then } -1 \textbf{ else abort}$$

When the widths of $\tilde{x}$ and $\tilde{\epsilon}$ get small enough, provided that $\epsilon \in \tilde{\epsilon}$ is positive, it succeeds eventually.

7.4 Fuzzy Wilkinson Shift

For a tridiagonal matrix T, the Wilkinson shift in Equation 7.1 can be directly obtained as follows [GVL96, § 8.3.3]:

$$\lambda_{\mathrm{WILK}} = T_{d,d} + \delta - \mathrm{sign}(\delta)\sqrt{\delta^2 + T_{d,d-1}^2}$$

where $\delta = (T_{d-1,d-1} - T_{d,d})/2$. The Wilkinson shift fails due to the computation of $\text{sign}(\delta)$ (which happens precisely when the shift candidates, two eigenvalues of the bottom-right 2×2 submatrix, are equally distanced from $T_{d,d}$). The natural relaxation is to make $\text{sign}(\delta)$ fuzzy thus that when δ is close to 0 with some formal tolerance factor, let the $\text{sign}(\delta)$ return any number among -1 and 1. The intuition is that when the distances between the last entry and the two eigenvalues are close enough, then it will not matter whichever is picked; nonetheless, there should be a quantitative criterion: what does it mean to be *close enough*? *Fuzzy* Wilkinson shift is the relaxed choice of the eigenvalue:

Definition 7.2. Let λ_1 and λ_2 be the eigenvalues of the bottom-right 2×2 submatrix $\begin{bmatrix} T_{d-1,d-1} & T_{d-1,d} \\ T_{d,d-1} & T_{d,d} \end{bmatrix}$ of an unreduced real symmetric tridiagonal matrix T. Fuzzy Wilkinson shift $\bar{\lambda}^{(\kappa)}_{\text{WILK}}$ with $0 < \kappa \leq 2^{-3}$ of the matrix is either λ_1 or λ_2 which satisfies the inequalities: $|T_{d,d} - \bar{\lambda}^{(\kappa)}_{\text{WILK}}| < |T_{d,d} - \lambda_i| + \kappa \cdot |T_{d,d-1}|$ for $i = 1, 2$. The shift $\bar{\lambda}^{(\kappa)}_{\text{WILK}}$ can be computed directly, with $\delta \coloneqq (T_{d-1,d-1} - T_{d,d})/2$, by the following formula:

$$\bar{\lambda}^{(\kappa)}_{\text{WILK}} = T_{d,d} - \overline{\text{sign}}\left(\delta, \kappa \cdot |T_{d,d-1}|\right) \sqrt{\delta^2 + T^2_{d,d-1}}. \tag{7.2}$$

Note that the fuzzy shift becomes more like the Wilkinson shift as $\kappa \to 0$. Fuzzy Wilkinson shift can be understood to be a *good* approximation to the exact Wilkinson shift relative to the factor $\kappa \cdot |T_{d,d-1}|$. The following lemma confirms that the relaxation is indeed safe:

Lemma 7.1. The QR algorithm on any real symmetric tridiagonal matrix T using the fuzzy Wilkinson shift converges. Moreover, when $\kappa = 2^{-5}$, the rate of convergence becomes $|T^{(j)}_{d,d-1}| < 2^{-j/2+2} \|T\|^2_F$ where $T^{(j)}$ denotes the tridiagonal matrix at j'th iteration.

7.5 Separating Eigenvalues

7.5.1 Interval Tridiagonal Reduction

Algorithm 2: interval_trig$(\tilde{A}, q)$

for $j = 1 \to d - 2$ **do**
  if $|\tilde{A}_{i,j}| < 2^{-q}$ *for all* $i > j$ **then**
    return $\{\tilde{A}_{1:j,1:j}\} \cup$ `interval_trig`$(\tilde{A}_{j+1:d,j+1:d}, q)$
  else if $|\tilde{A}_{i,j}| > 2^{-q-1}$ *for some* $i > j$ **then**
    proceed
  else
    abort
  end
  $\tilde{s}_1 \coloneqq \sqrt{\sum_{i=j+1}^{d} (\tilde{A}_{i,j})^2}$ and $\tilde{s}_2 \coloneqq \tilde{s}_1 + \text{sign}_T(\tilde{A}_{j+1,j}) \cdot \tilde{A}_{j+1,j}$;
  $\tilde{A}_{j+1:d,j} \coloneqq [\tilde{s}_1, 0, \cdots, 0]$ and $\tilde{A}_{j,j+1:d} \coloneqq [\tilde{s}_1, 0, \cdots, 0]^T$;
  $\tilde{u}_1 \coloneqq [\text{sign}_T(\tilde{A}_{j+1,j}) + \tilde{A}_{j+1,j} \,/\, \tilde{s}_1, \tilde{A}_{j+2,j} \,/\, \tilde{s}_1, \cdots, \tilde{A}_{d,j} \,/\, \tilde{s}_1]$;
  $\tilde{u}_2 \coloneqq [1, \tilde{A}_{j+2,j} \,/\, \tilde{s}_2, \cdots, \tilde{A}_{d,j} \,/\, \tilde{s}_2]$;
  $\tilde{A}_{j+1:d,j+1:d} \coloneqq \tilde{A}_{j+1:d,j+1:d} - \tilde{u}_1 * (\tilde{u}_2^T * \tilde{A}_{j+1:d,j+1:d})$;
  $\tilde{A}_{j+1:d,j+1:d} \coloneqq \tilde{A}_{j+1:d,j+1:d} - (\tilde{A}_{j+1:d,j+1:d} * \tilde{u}_2) * \tilde{u}_1^T$
end
return $\{\tilde{A}\}$

A symmetric matrix is efficiently reduced to a tridiagonal matrix via a similarity transformations using Householder reflections [GVL96, § 8.3.1]. For any index i, define a householder reflector $H = I - 2uu^T$ where u is the normalized vector of $[A_{i+1,i} + \mathrm{sign}(A_{i+1,i}) \left\|A_{i:d,i}\right\|_2, A_{i+2,i}, \cdots, A_{d,i}]$ with regards to the 2-norm. Then, the similarity transformation $A' \coloneqq \begin{bmatrix} I & 0 \\ 0 & H \end{bmatrix} A \begin{bmatrix} I & 0 \\ 0 & H \end{bmatrix}$ satisfies $A'_{i,j} = A'_{j,i} = 0$ for all $j > i + 1$. Repeating this from $i = 1$ to $i = d - 2$, we get a similar tridiagonal matrix. We explain how the interval variant (Algoritm 2) differs.

Line 2-3: In the classical algorithm, for an index i, when the entries below (i, i) are all 0, it is instructed to split the matrix into the two submatrices $A_{1:i,1:i}$ and $A_{i+1:d,i+1:d}$. It not only reduces the complexity of the eigenvalue finding but also ensures that each input matrix to the QR algorithm does not have repeated eigenvalues. However, when the entries are intervals, we cannot decide if the ideal real number represented by an interval is identical to zero. Thereby, we require an additional parameter $q \in \mathbb{N}$, instead of checking if the entries are *precisely* zero, we test if the magnitudes of the intervals are bounded above by 2^{-q}. More precisely, if $|\tilde{A}_{j,i}| < 2^{-q}$ for all $j > i$, the we simply regard $\tilde{A}_{j,j+1:d} = \tilde{A}^T_{j+1:d,j} = 0$, and split the matrix. The consequence of this neglecting is analyzed later.

Line 4-8: Additionally, in order to bound the width growths of divisions in the construction of Householder reflectors, the algorithm proceeds only when there is an entry that is bounded away from zero by 2^{-q-1}. Note that when the widths of the interval entries in a column are less than 2^{-q-1}, it can be decided either an entry's absolute interval is bounded above by 2^{-q} or bounded below by 2^{-q-1}. Hence, Line 2-9 succeeds (does not abort) when the widths of the interval entries are less than 2^{-q-1}.

Line 9-14: The $\mathrm{sign}(A_{i+1,i})$ in the construction of the Householder reflector is only for numerical stability. Hence, if we cannot decide the sign of the interval entry, when $\tilde{A}_{i+1,i}$ contains 0, we let it be -1. The interval operator sign_T in Line 9,11 denotes the operation; i.e., $\mathrm{sign}_T(\tilde{x}) \equiv \textbf{if } \tilde{x} > 0 \textbf{ then } 1 \textbf{ else } -1$. See that $u_1 u_2^T$ in Line 13-14 is $2uu^T$.

Note that neglecting only happens when $|\tilde{A}_{i,j}| < 2^{-q}$ for all $i > j$ in j'th iteration. Neglecting is the operation of perturbing a matrix $A \in \tilde{A}$ by some P where $\|P\|_1 \leq d \cdot 2^{-q}$. For a symmetric A, Weyl's inequality ensures that its eigenvalues get perturbed by at most $d^2 \cdot 2^{-q}$, considering that there can be at most d perturbations. Therefore, reducing $A \in \tilde{A}$ to a tridiagonal matrix with this splitting strategy will result in interval matrices $\{\tilde{T}_1, \cdots, \tilde{T}_m\}$ each containing an *unreduced* tridiagonal matrix $T_i \in \tilde{T}_i$ whose eigenvalues approximate the eigenvalues of the original matrix as follows. Each eigenvalue of A is contained in at least one interval of $\cup_i \{[\lambda - d^2 \cdot 2^{-q}, \lambda + d^2 \cdot 2^{-q}] \ : \ \lambda \in \Lambda(T_i)\}$.

Lemma 7.2. Consider a symmetric matrix A and an interval matrix $\tilde{A}$ such that $A \in \tilde{A} \in \mathbb{ID}_n^{d \times d}$. Algorithm interval_trig succeeds and produces a list of interval matrices whose entries' widths are bounded by 2^{-p} when $w(\tilde{A}) < 2^{-m(A,q,p)}$ and $q > |\log \|A\|_F|$ where $m(A, q, p) \in p + \mathcal{O}\big(d(q + \log d + \log \|A\|_F)\big)$. Under this condition, including that n should be large enough to enable $w(\tilde{A}) < 2^{-m(A,q,p)}$, the number of bit operations is bounded by $C(A, n) \in \mathcal{O}\big(d^3 \cdot \mathtt{M}(\log d + \log \|A\|_F + n)\big)$

Proof. (Sketch) We need to analyze how much the interval widths grow in each iteration. Suppose at the beginning of an iteration, there is an off-diagonal entry bounded away from zero by 2^{-q-1} (otherwise, the iteration is skipped anyway), and the widths of all entries are smaller than 2^{-q-2}. The square sum at Line 9 has its width bounded by $1/2$ and is bounded away from zero by 2^{-q-1}. Bounding the denominator away from zero, we can bound the width and magnitude growths in that the magnitudes of $\tilde{u}_1, \tilde{u}_2$ are bounded above by 4 and their widths are bounded above by $w 2^{\mathcal{O}(q + \log(md))}$. Here, m is a magnitude bound, and w is a width bound of $\tilde{A}$ at the beginning of the iteration. Further doing a tedious calculation, we get that at the end of the iteration, the width of $\tilde{A}$ is bounded by $w 2^{\mathcal{O}(q + \log(mw))}$.

See that $m \leq \|A\|_F + w < 2\|A\|_F$ due to $w < 2^{-q-2}$ and $q > |\log\|A\|_F|$. Hence, when the initial width w is smaller than $2^{-\mathcal{O}(q\log(d\|A\|_F)}$, the widths of the entries in the next iteration again satisfies the condition being smaller than 2^{-q-2}. Therefore, if the initial intervals' widths are smaller than $2^{-\mathcal{O}(q\log(d\|A\|_F)}$. And, when the initial widths are smaller than $2^{-p-\mathcal{O}(q\log(d\|A\|_F)}$, the final widths are smaller than 2^{-p}.

Since the Frobenius norm is rotation invariant, at the beginning and the end of each iteration, the magnitudes of entries are bounded by $2\|A\|_F$.

The bit-cost is bounded by the $\mathcal{O}(d^3)$ multiplications in Line 13-14 on intervals whose magnitudes are bounded by $\mathcal{O}(d\,\|A\|_F)$. □

7.5.2 Interval QR Step with Fuzzy Shift

Lemma 7.1 guarantees that the QR algorithm using the relaxed *fuzzy* Wilkinson shift converges. The fuzzy Wilkinson shift can be applied to the interval version of QR algorithm.

Algorithm 3: QR_step($\tilde{T}$)

$\tilde{\omega} \coloneqq \tilde{T}_{d,d} + \tilde{\delta} - \underline{\mathrm{sign}}\left(\tilde{\delta}, 2^{-7} * \left|\tilde{\beta}_{d-1}\right|\right) * \sqrt{\tilde{\delta}^2 + \tilde{\beta}_{d-1}^2}$;
for $i := 1 \to d$ **do** $\tilde{T}_{j,j} \coloneqq \tilde{T}_{j,j} - \tilde{\omega}$;
for $j := 1 \to d-1$ **do**
  $\tilde{G}_j \coloneqq \begin{bmatrix} \tilde{T}_{j,j} & \tilde{T}_{j+1,j} \\ -\tilde{T}_{j+1,j} & \tilde{T}_{j,j} \end{bmatrix} / \sqrt{\tilde{T}_{j,j}^2 + \tilde{T}_{j+1,j}^2}$;
  $\tilde{T}_{j:j+1,\max(1,j-1):\min(d,j+1)} \coloneqq \tilde{G}_j * \tilde{T}_{j:j+1,\max(1,j-1):\min(d,j+1)}$
end
for $j := 1 \to d-1$ **do**
  $\tilde{T}_{\max(1,j-1):\min(d,j+1),j:j+1} \coloneqq \tilde{T}_{\max(1,j-1):\min(d,j+1),j:j+1} * \tilde{G}_j^T$
end
for $i := 1 \to d$ **do** $\tilde{T}_{j,j} \coloneqq \tilde{T}_{j,j} + \tilde{\omega}$;
return $\tilde{T}$

Here, $\tilde{\beta}_{d-1} \coloneqq \tilde{T}_{d,d-1}$ and $\tilde{\delta} \coloneqq \left(\tilde{T}_{d-1,d-1} - \tilde{T}_{d,d}\right)/2$. In Line 2, we shift the input matrix with the interval fuzzy Wilkinson shift that is computed in Line 1. In Line 3-9, we apply implicit QR step using Givens rotations. (See [GVL96, § 8.3.3].) In Line 10, we shift the resulting matrix back. The off-diagonal terms of the input interval matrix are expected to be bounded away from zero by 2^{-q-1}. The QR algorithm introduced later, which repeatedly calls the interval QR step will only feed those matrices. The condition which makes Algorithm 3 correct can be analyzed as follows:

Lemma 7.3. Consider an unreduced symmetric tridiagonal matrix T and an interval matrix $\tilde{T}$ such that $T \in \tilde{T} \in \mathbb{ID}_n^{d\times d}$. Suppose the off-diagonal entries of $\tilde{T}$ are bounded away from zero by 2^{-q-1}. Then, Algorithm QR_step on $\tilde{T}$ succeeds and returns an interval matrix whose entries' widths are bounded by 2^{-p} when $w(\tilde{T}) < 2^{-m(T,q,p)}$ and $q > |\log\|T\|_F|$ where $m(T,q,p) \in p + \mathcal{O}\big(d(q + \log d + \log\|A\|_F)\big)$. Under this condition, including that n should be large enough to enable $w(\tilde{T}) < 2^{-m(T,q,p)}$, the number of bit operations is bounded by $C(T,q,n) \in \mathcal{O}(d \cdot \mathtt{M}(\log d + \log\|T\|_F + n))$.

Proof. The fuzzy Wilkinson shift of the tridiagonal T is bounded by $\|T\|_F$. Hence, after shifting, $\|T - \omega_\kappa I\|_F \leq \|T\|_F + \sqrt{d}\|T\|_F < 4\sqrt{d}\|T\|_F$.

Let w be the width of the interval matrix at the beginning of the first iteration of Line 4. Suppose $w < 2^{-q-2}$. Then, the magnitude is bounded by $4\sqrt{d}\|T\|_F$ similarly to the proof of Lemma 2. The denominator being bounded away from zero by 2^{-q-1} ensures that the widths of $\tilde{G}_1$ is less than $w2^{\mathcal{O}(q+\log\|T\|_F+\log d)}$. And, after the rotation by $\tilde{G}_1$, the width growth is bounded by $w' < w2^{\mathcal{O}(q+\log\|T\|_F+\log d)}$. Hence, repeating this for $2(d-1)$ iterations, when $w < 2^{-m(T,q,p)}$, we get the resulting intervals' widths bounded by 2^{-p}.

The overall bit-cost is bounded by $\mathcal{O}(d)$ integer multiplications on operands whose magnitudes are bounded by $\mathcal{O}(\sqrt{d}\,\|T\|_F)$. □

7.5.3 Interval QR Algorithm Fuzzy Shift

Algorithm 4: interval_QR$(\tilde{T}, q)$

```
if d = 1 then return {T̃_{1,1}} ;
for j = 1 → d − 1 do
    if T̃_{j,j+1} < 2^{−q} then
        return interval_QR(T̃_{1:j,1:j}, q) ∪ interval_QR(T̃_{j+1:d,j+1:d}, q)
    else if T̃_{j,j+1} > 2^{−q−1} then
        proceed
    else
        abort
    end
end
return interval_QR(QR_step(T̃), q)
```

Using the parameter q, in Line 2-10, the interval QR algorithm inspects the input interval matrix going through all off-diagonal entries. If there is an index j such that $|\tilde{T}_{j+1,j}| < 2^{-q}$ holds, it split the matrix and recursively feeds the two matrices $\tilde{T}_{1:j,1:j}$ and $\tilde{T}_{j+1:d,j+1:d}$ to the interval QR algorithm; see Line 3-4. Otherwise, only when all off-diagonal entries are bounded away from zero by 2^{-q-1}, the QR step applied; see Line 5-9. Note that when the widths of the interval entries are smaller than 2^{-q-1}, at least one of the two holds; i.e., Line 8 will not be reached in that case.

This splitting is equivalent to a perturbation by a matrix P which has only the two possibly nonzero entries which are $|P_{j+1,j}| = |P_{j,j+1}| < 2^{-q}$. Since $\|P\|_1 < 2^{-q}$, deflation causes eigenvalues to be perturbed by at most 2^{-q}. The interval QR algorithm goes on until it deflates all the off-diagonal entries.

Let $T \in \tilde{T}$. Lemma 7.1 guarantees that when the QR algorithm is applied to T with fuzzy Wilkinson shift, $q+2\log\|T\|_F+1$ iterations are enough in order to ensure $|T_{d,d-1}| < 2^{-q-1}$. When $|T_{d,d-1}| < 2^{-q-1}$ and $w(\tilde{T}_{d,d-1}) < 2^{-q-1}$ both hold, deflation happens as $\tilde{T}_{d,d-1} < 2^{-q}$. Since deflation can only decrease $\|T\|_F$ and $\|T\|_F$ is preserved throughout the QR steps, the total number of iterations is bounded by $d\cdot(q+2\log\|T\|_F+1)$ assuming that the widths stay tight enough. When the preconditions in Lemma 7.3 hold throughout the iterations, the interval QR steps do not fail throughout the iterations. Hence, the interval QR algorithm succeeds:

Lemma 7.4. Consider an unreduced symmetric tridiagonal matrix T and an interval matrix $\tilde{T}$ such that $T \in \tilde{T} \in \mathbb{ID}_n^{d\times d}$. Algorithm interval_QR on $(\tilde{T}, q)$ succeeds and returns a list of intervals whose widths are bounded by 2^{-p} when $w(\tilde{T}) < 2^{-m(T,q,p)}$ and $q > |\log\|T\|_F|$ where $m(T,q,p) \in p + \mathcal{O}\big(d^2(q+$

$\log d + \log \|T\|_F)(q + \log \|T\|_F)\big)$. Under this condition, including that n should be large enough to express $w(\tilde{T}) < 2^{-m(T,q,p)}$, the number of bit operations is bounded by $C(T,q,n) \in \mathcal{O}(d^2(q+\log\|T\|_F)\cdot \mathtt{M}(\log d + \log\|T\|_F + n))$.

7.5.4 Separating Eigenvalues

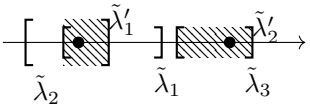


Figure 7.1: Classifying on $\tilde{\lambda}_1, \tilde{\lambda}_2$ and $\tilde{\lambda}_3$, the three enlarged intervals, yields the pairs $(\tilde{\lambda}'_1, 2)$ and $(\tilde{\lambda}'_2, 1)$ where $\tilde{\lambda}'_1 \coloneqq \tilde{\lambda}_1 \cap \tilde{\lambda}_2$. If the distance between the actual eigenvalues (the two filled circles) is greater than some ϵ and the widths of the intervals are less than $\epsilon/2$, the procedure succeeds.

Applying Algorithm 2 to an interval matrix $\tilde{A}$, which contains a symmetric matrix A with k distinct eigenvalues, results in a list of interval matrices. Further application of Algorithm 4 to each interval matrix yields a list of intervals $(\tilde{I}_i)_{i=1,\cdots,d}$ which approximate the eigenvalues of A; according to the bounds we obtained, each of the expanded intervals $(\tilde{\lambda}_i)_{i=1,\cdots,d}$, where $\tilde{\lambda}_i \coloneqq \tilde{I}_i + [-(d^2+d)\cdot 2^{-q}, (d^2+d)\cdot 2^{-q}]$, contains each eigenvalue of A respecting multiplicities. It remains to classify $(\tilde{\lambda}_i)$ into distinct eigenvalues of A: let us name $\mathsf{filter}(q, k, \tilde{I}_1, \cdots, \tilde{I}_n)$ to be the procedure that (1) enlarges all the intervals by $(d^2+d)\cdot 2^{-q}$, (2) when there are intersecting intervals, keeps the intersection and counts the number of the intersecting intervals, and after that, (3) if there are exactly k disjoint intervals, returns the list of pairs of an interval and the corresponding multiplicity; otherwise, it aborts. Note that the classification does not increase the widths of intervals; see Figure 7.1. Combining all the contents in this section yields the following algorithm to classify all the distinct eigenvalues:

Algorithm 5: $\mathsf{separate\_eig}(\tilde{A}, k)$

try with $q \coloneqq 2^0, 2^1, \cdots, 2^{\lfloor \frac{1}{2}\log\log 1/w(\tilde{A}) - \log d\rfloor}$ **in**
  $\{\tilde{T}_1, \cdots, \tilde{T}_m\} \coloneqq \mathsf{interval\_trig}(A, q)$;
  for $j \coloneqq 1 \to m$ **do** $\Lambda_j \coloneqq \mathsf{interval\_QR}(\tilde{T}_j, q)$;
  $\{(\tilde{\lambda}_1, \mu_1), \cdots, (\tilde{\lambda}_k, \mu_k)\} \coloneqq \mathsf{filter}(q, k, \Lambda_1, \cdots, \Lambda_m)$
end
return $\{(\tilde{\lambda}_1, \mu_1), \cdots, (\tilde{\lambda}_k, \mu_k)\}$

The algorithm computes the interval eigenvalues with increasing q. Even when we have intervals that approximate the eigenvalues (in Line 3), we need to expand those with proportional to 2^{-q} and classify k disjoint intervals among those (in Line 4). When q is not small enough to separate distinct eigenvalues, it fails in Line 4. Hence, we repeatedly run the overall procedure with increasing q in an exponential increment.

Proof of Theorem 7.2-1. Let $\tilde{A}$ be an input that contains A and suppose $w(\tilde{A}) < 2^{-m}$ for some m. Taking $p \coloneqq q + 2\log d + 1$ in Lemma 7.4 yields that when $w(\tilde{T}_i) < 2^{-\mathcal{O}\left(d^2(q+\log d+\log\|T\|_F)^2\right)}$, the widths of the intervals computed by Line 3 are bounded by $2d^2 2^{-q}$. Hence, the intervals after enlarging at Line 4 are bounded by $2^{-q+2\log d+3}$. Lemma 7.2 implies that the assumption can be met when $m \in \mathcal{O}\big(d^2(q + \log d + \log\|A\|_F)^2\big)$.

Therefore, in the case where $m \in \mathcal{O}\big(d^2(p + \log d + \log \|A\|_F)^2\big)$ and q reaches $\sqrt{m}/d \simeq \mathcal{O}\big(p + \log d + \log \|A\|_F)\big)$, the return intervals' widths are bounded by 2^{-p}.

See that Line 4 of classifying intervals succeeds when $2^{-q+2\log d+2} < \Delta_\lambda$ where $\Delta_\lambda \coloneqq \Delta(A) \cdot \|A\|_F$; hence, when $q > 2\log d + \log 1/\Delta_\lambda + 2$ and $q > |\log \|A\|_F|$, the procedure succeeds which is the case when $q > \mathcal{O}(\log d + \log 1/\Delta + |\log \|A\|_F|)$. Therefore, when $m \in \mathcal{O}\big(d^2(p + \log 1/\Delta + \log d + |\log \|A\|_F|)^2\big)$, the overall procedure succeeds and returns disjoint intervals whose widths are bounded by 2^{-p}. Since the cost is polynomial in q, with the exponential increments of q, the last iteration with $q = \sqrt{m}/d$ dominates. □

7.6 Interval Kernel Problem

This section considers the interval eigenspace problem, which can be solved by computing a basis of the kernel of a singular interval matrix: when an interval matrix which contains a singular matrix is given, it is required to compute a set of interval vectors where each contains such a vector that the vectors span the singular matrix's kernel.

7.6.1 Interval Gaussian Algorithm

We consider an interval Gaussian algorithm to solve the problem. The classical Gaussian algorithm on real numbers conventionally searches for a pivot element whose magnitude is the greatest. In order to achieve the continuity property (iii), we consider complete pivot searching that searches for a pivot element whose magnitude is greater than the half of the greatest magnitude [BCK+16, § 2.7]; see that when the widths of intervals approximating the real entries are small enough, eventually, it succeeds in locating an interval pivot element. It is fuzzy in the sense that when there are multiple such entries, any of those can be chosen to be a pivot.

It is well known that an interval Gaussian algorithm does not preserve the regularity of a matrix and indeed this is the case for the above interval Gaussian algorithm that the rank of an interval matrix will not be preserved throughout the iterations; e.g., see the famous example of [Rei79]. Algorithm 6 on $(\tilde{A}, k)$, where $\tilde{A} \ni A$ and $\operatorname{rank}(A) = k$, is correct if Phase 1 runs all its k iterations.

Throughout the iterations in Phase 1, the interval matrices always contain a matrix whose kernel is $\ker(A)$ and whose rank is k. Hence, throughout the iterations of Phase 1, there always is an interval entry in the submatrix where the pivot searching happens containing a nonzero number r; otherwise, the kernel of A is not preserved. Note that when the widths of the interval entries are sufficiently small, the interval containing r can be picked as a pivot element. This gives evidence that success depends on the widths of the interval entries which is bounded by Lemma 7.5.

Lemma 7.5.

1. Let $(\tilde{A}, k)$ be an input to Algorithm 6. Then, the widths of the entries of the interval matrices constructed throughout Phase 1 are bounded by $60^k(1 + m(\tilde{A}))w(\tilde{A})$ and the magnitudes of the entries are bounded by $5^k m(\tilde{A})$.

2. Consider a variant algorithm of Algorithm 6 which works on real intervals using standard interval computations in [AH84] instead. Then, the entries of the interval matrices, constructed throughout the iterations in Phase 1, of the variant algorithm have widths bounded by $11^k w(\tilde{A})$.

Algorithm 6: interval_gaussian$(\tilde{A}, k)$

for $j \coloneqq 1 \to k$ **do** `// Phase 1`
  compute $b \coloneqq \max_{j \le \ell, m \le d} m(\tilde{A}_{\ell,m})$;
  find (ℓ, m) s.t. $|\tilde{A}_{\ell,m}| \ > \ b/2$ for $j \le \ell, m \le d$ **abort** if fail;
  swap rows indexed j and ℓ and **swap** columns indexed j and m;
  record the column permutations in Π;
  for $\ell \coloneqq j+1 \to d$ **do** $\tilde{s}_\ell \coloneqq \tilde{A}_{j,\ell} \ / \ \tilde{A}_{j,j}$;
  for $\ell \coloneqq j+1 \to d$ ***and*** $m \coloneqq j \to d$ **do**
    if $m = j$ **then** $\tilde{A}_{\ell,m} \coloneqq 0$ **else** $\tilde{A}_{\ell,m} \coloneqq \tilde{A}_{\ell,m} \ - \ \tilde{A}_{\ell,j} \ * \ \tilde{s}_\ell$;
  end
end
for $i \coloneqq 1 \to k$ **do** `// Phase 2`
  $\tilde{s}_i \coloneqq \tilde{A}_{i,i}$;
  for $j \coloneqq i \to d$ **do if** $i = j$ **then** $\tilde{A}_{i,j} \coloneqq 1$ **else** $\tilde{A}_{i,j} \coloneqq \tilde{A}_{i,j} \ / \ \tilde{s}_i$;
  ;
end
for $j \coloneqq 2 \to k$ ***and*** $\ell \coloneqq 1 \to j-1$ **do**
  $\tilde{s}_\ell \coloneqq \tilde{A}_{\ell,j}$;
  for $m \coloneqq j \to d$ **do** $\tilde{A}_{\ell,m} \coloneqq \tilde{A}_{\ell,m} \ - \ \tilde{A}_{j,m} \ * \ \tilde{s}_\ell$;
end
for $i \coloneqq k+1 \to d$ ***and*** $j \coloneqq k+1 \to d$ **do**
  if $i = j$ **then** $\tilde{A}_{i,j} \coloneqq -1$ **else** $\tilde{A}_{i,j} \coloneqq 0$;
end
return $(\Pi \cdot \tilde{A})_{1 \le i < d, k < j \le d}$

We defer the proof of this lemma to the appendix. The bounds are obtainable mainly because we force the pivots to be bounded away from zero by $b/2$; the interval over-estimations are bounded accordingly.

7.6.2 Pseudo-regularity

Consider the case when the algorithm is applied to an interval matrix $\tilde{A}$ which contains A and whose rank is $k = \mathrm{rank}(A)$. Suppose that the algorithm succeeds and yields the k pivot elements $\tilde{p}_1, \cdots, \tilde{p}_k$ at the end of the first phase. Then, there are nonzero real numbers $p_1, \cdots, p_k$ such that $p_i \in \tilde{p}_i$ for all i and $\Pi_i |p_i|$ is the magnitude of a nonzero k'th principal minor of A. Now, let us take a look at the pivoting strategy.

Suppose p_i is chosen to be the pivot element at i'th iteration, and b_i is the greatest magnitude of all entries in the being-searched submatrix. Note that $|b_i|/2 < |p_i| \le |b_i|$ holds for all i. After eliminating all rows below by the row led by p_i, the magnitudes of the entries in the resulting submatrix are bounded above by $3|b_i|$.

Since $|b_i|/2 < |p_i|$ holds, it holds that $|p_{i+1}| \le 3|b_i| < 6|p_i|$. Due to $\|A\|_{\max}/2 < |p_1| \le \|A\|_{\max}$, we have $|p_i| \le 6^{i-1} \|A\|_{\max}$ for all i. Therefore, if m is the smallest magnitude of nonzero k'th principal minors of A, the inequality $m < \Pi_i |p_i| < |p_i| 6^{0+\cdots+(i-1)+(i+1)+\cdots+k-1} \|A\|_{\max}^{k-1}$ holds for all i.

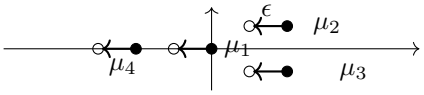


Figure 7.2: Changes in eigenvalues to the perturbation with ϵ in the proof of Lemma 7.6. Shifting with small enough ϵ on a singular diagonalizable matrix makes it regular and its eigenvalues are shifted by ϵ.

Thus, the magnitudes of the pivot elements of A are bounded from below by $m \cdot \|A\|_{\max}^{-k+1} \cdot 6^{k(k-1)/2}$, which means, throughout the iterations in the interval Gaussian algorithm, the computed maximum magnitudes are greater than or equal to the bound. Together with Lemma 7.5, a condition for success can be obtained related to the quantity m.

However, though the quantity m catches the notion of pseudo-regularity and the intuition that the condition for success depends on it, it is not satisfying as it does not have a clear connection to the geometric distribution of eigenvalues which we used as a parameter in the eigenvalue computation. Instead, we propose the quantitative measure:

Definition 7.3. For a diagonalizable real matrix A with rank k, let $|\lambda|_{\min}(A) \coloneqq \min\{|\lambda| \in \Lambda(A)\}$ be the minimum nonzero magnitude of its eigenvalues. Then, the *pseudo-regularity*, a quantitive measure saying how far away the matrix A is from having rank less than k, is defined to be $\delta_\lambda(A) \coloneqq |\lambda|_{\min}(A) / \|A\|_F$.

Let A be a diagonalizable real matrix whose rank is k. Consider $A' \coloneqq A - \epsilon \cdot I$ to be a perturbation on A by $\epsilon \in \mathbb{R}$; see Figure 7.2. When ϵ is in $(0, |\lambda|_{\min}(A))$, the perturbed matrix A' is regular. Consider an interval matrix $\tilde{A} \coloneqq [A', A] \in \mathbb{IR}^{d \times d}$. If ϵ is small enough, then Phase 1 of the variant interval Gaussian algorithm, which works on $\mathbb{IR}$, on $\tilde{A}$ will result in an interval matrix which contains H and H' such that $\ker H = \ker A$ and $\ker H' = \ker A'$:

$$H = \begin{bmatrix} p_1 & \cdots & * & * & \cdots & * \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & p_k & * & \cdots & * \\ 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 \end{bmatrix} \quad H' = \begin{bmatrix} p'_1 & \cdots & * & * & \cdots & * \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & p'_k & * & \cdots & * \\ 0 & \cdots & 0 & \epsilon_{1,1} & \cdots & \epsilon_{1,n-k} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \epsilon_{n-k,1} & \cdots & \epsilon_{n-k,n-k} \end{bmatrix}$$

By the second part of Lemma 7.5, the entry-wise difference between two matrices is less than $11^k\epsilon$. Therefore, $|\epsilon_{i,j}| < 11^k\epsilon$. By Hadamard's bound, it holds that $|\det(H')| < |p'_1 p'_2 \cdots p_k|' \cdot (11^k\epsilon)^{d-k} \cdot (d-k)^{(d-k)/2}$. On the other hand, as $|\det(H')| = |\det(A+\epsilon)| = \prod |\lambda_i - \epsilon| > \epsilon^{d-k} \cdot (|\lambda|_{\min} - \epsilon)^k$, the following inequality is obtained:

$$\begin{aligned} \epsilon^{d-k} \cdot (|\lambda|_{\min} - \epsilon)^k &< |p'_1 p'_2 \cdots p'_k| \cdot (11^k\epsilon)^{d-k} \cdot (d-k)^{(d-k)/2} \\ &< (|p_1| + 11^k\epsilon) \cdots (|p_k| + 11^k\epsilon) \cdot (11^k\epsilon)^{d-k} \cdot (d-k)^{(d-k)/2} \end{aligned}$$

Therefore, the bound $(|\lambda|_{\min} - \epsilon)^k < (|p_1| + 11^k\epsilon) \cdots (|p_k| + 11^k\epsilon) \cdot 11^{k(d-k)} \cdot (d-k)^{(d-k)/2}$ holds for any small enough ϵ. Considering that the algorithm only can make pivot values grow by a factor of at most 6 in each iteration, $|p_i| \leq 6^{i-1} \|A\|_{\max} \leq 6^{i-1} \|A\|_F$ holds. Hence, $|p_1 \cdots p_k| \leq p_j \cdot 6^{k(k-1)/2} \cdot \|A\|_F^{k-1}$ for any j. Therefore, for any j, the following holds:

$$|\lambda|_{\min}^k < |p_1 p_2 \cdots p_k| \cdot 11^{k(d-k)} \cdot (d-k)^{(d-k)/2}$$

$$\leq p_j \cdot 6^{k(k-1)/2} \|A\|_F^{k-1} \cdot 11^{k(d-k)} \cdot (d-k)^{(d-k)/2}$$

Since any choice of a pivot element made by Algorithm 6 is valid in the variant algorithm, an algorithm in $\mathbb{IR}$, the inequality is still valid for Algorithm 6:

Lemma 7.6. Consider a diagonalizable matrix $A \in \mathbb{R}^{d\times d}$ with $k \coloneqq \operatorname{rank}(A)$. For any interval matrix containing A when the interval Gaussian algorithm searches for a pivot element, there always exists an entry whose absolute interval contains a real number greater than $2^{-4d^2}\delta_\lambda^k(A)\,\|A\|_F$.

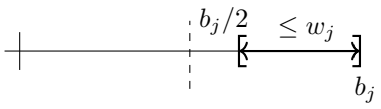


Figure 7.3: When the width w_j is smaller than $b_j/2$, the interval can be selected as a pivot element.

This gives a straightforward condition succeeding on finding pivot element. The widths of intervals staying smaller than $2^{-4d^2}\delta_\lambda^k(A)\,\|A\|_F$ throughout the iterations:

Lemma 7.7. Consider a diagonalizable matrix A whose rank is k and an interval matrix $\tilde{A}$ such that $A \in \tilde{A} \in \mathbb{ID}_n^{d\times d}$. Algorithm interval_gaussian on $(\tilde{A}, k)$ succeeds and returns interval vectors whose entries' widths are bounded by 2^{-p} when $w(\tilde{A}) < 2^{-m(p,A)}$ where $m(p,A) \in p+\mathcal{O}(d^2+d\log 1/\delta_\lambda(A)+|\log\|A\|_F|)$. Under this condition, including that n should be large enough to enable $w(\tilde{A}) < 2^{-m(p,A)}$, the number of bit operations is bounded by $\mathcal{O}(d^3 \cdot \mathtt{M}(d+n+\log\|A\|_F))$.

Proof. Lemma 7.6 says that throughout the iterations in Phase 1, when a pivot element is searched, there is an interval $\tilde{b}_j$ whose magnitude is the computed maximum magnitude in Line 2 such that $b_j \coloneqq m(\tilde{b}_j) > 2^{-4d^2}\delta_\lambda^k(A)\,\|A\|_F$. If E is an upper bound on the widths of the interval entries, then if $E < b/2$ the interval $\tilde{b}$ can be chosen as a pivot element; see Figure 7.3. The overall bit-cost is bounded by $\mathcal{O}(d^3)$ multiplications on operands whose magnitudes are less than $\mathcal{O}(5^d\|A\|_F)$ from Lemma 7.5. □

Now, we can give the proof of the second part of our main result:

Proof of Theorem 7.2-2. We run the Gaussian algorithm on the interval matrix $\tilde{A}-\tilde{\lambda}$ where $\tilde{A}$ contains a diagonalizable A and $\tilde{\lambda}$ contains an eigenvalue λ of A. See that $\log 1/\delta_\lambda(A) = \log\|A-\lambda\|_F/|\lambda|_{\min}(A-\lambda) = \log\|A-\lambda\|_F/\|A\|_F + \log\|A\|_F/|\lambda|_{\min}(A-\lambda) < \log(1+\sqrt{d}) + \log 1/\Delta(A)$ where $\Delta(A)$ is the smallest distance among distinct eigenvalues of A relative to $\|A\|_F$. Using the bounds obtained in Lemma 7.7 yields the desired result. □

Chapter 8. Conclusion

In the earlier part of this dissertation, we have defined an imperative language that supports the functionality of real number computation based on computable analysis. The foundation of the language is sound that for any real number expression e, if the expression is well-defined, the real number expression really does evaluate to the number that the expression mathematically represents. In consequence, the users of the language can program real number computation assuming real numbers as abstract mathematical entities, and reason on the behaviours of their programs relying on their mathematical knowledge without considering artificial roundoff errors.

We wanted the languages to be imperative not only because imperative programming is commonly used paradigm in scientific computing practices. Imperative programming admits well-studied precondition-postcondition-style program specification and Hoare-style program verification methodologies. In addition to defining the formal semantics of the language, in this dissertation, we also devised the Hoare-style verification calculus that can be used to formally verify programs written in the languages. Consequently, the users of the language can easily specify the expected behaviours of their programs and either prove of disprove the correctness of the specifications such that verified computation over real numbers is obtained conveniently.

We suggested a way to formally extend the language with other continuous data such as real matrices and continuous real functions alongside with extending the verification calculus. However, there is a limitation that the language itself does not support construction of higher-oder objects. For example, we could program a functional that finds the root of input continuous real functions. However, it is not possible to construct arbitrary functions within a program. This leads to a natural and essential future work that is to extend the language with general higher-order data that often appear in scientific computing such as Sobolev functions, analytic functions, operators, functionals, and so on [KST18, TKZ18, SS17, Col20]. Of course, the goal is also to extend the verification calculus accordingly [DJ83, YHB07]

In the later part of this dissertation, when an interval matrix $\tilde{A}$ and a natural number k are given with a promise that $\tilde{A}$ contains a symmetric matrix A which has k distinct eigenvalues, we consider the computational problem of approximating all the eigenvalues and associated eigenspace of A using interval computations. We devised an interval QR algorithm and used an interval Gaussian algorithm to solve the problem and analyzed a condition that guarantees the computation to produce intervals whose widths are bounded by 2^{-p}, for any natural number p. The condition is parametrized by the relative eigenvalue separation of A, which we suggest to be a natural parameter of the problem. Having the explicit condition, an analysis is applied to obtain a bound for the bit-cost of the matrix eigenproblem for degenerate real symmetric matrices for reliable computing.

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[ABS18] Sewon Park Andrej Bauer and Alex Simpson. Command-like Expressions for Real Infinite-precision Calculations. *Dagstuhl Reports (Dagstuhl Seminar 17481)*, 7(11):149–150, 2018.

[AH84] Gotz Alefeld and Jurgen Herzberger. Introduction to interval computation. 1984.

[AO19] Krzysztof R Apt and Ernst-Rüdiger Olderog. Fifty years of hoare's logic. *Formal Aspects of Computing*, 31(6):751–807, 2019.

[AP86] Krzysztof R Apt and Gordon D Plotkin. Countable nondeterminism and random assignment. *Journal of the ACM (JACM)*, 33(4):724–767, 1986.

[Apt81] Krzysztof R Apt. Ten years of hoare's logic: A survey—part i. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 3(4):431–483, 1981.

[Apt83] Krzysztof R Apt. Ten years of hoare's logic: A survey—part ii: Nondeterminism. *Theoretical Computer Science*, 28(1-2):83–109, 1983.

[AY07] Jeremy Avigad and Yimu Yin. Quantifier elimination for the reals with a predicate for the powers of two. *Theor. Comput. Sci.*, 370(1-3):48–59, 2007.

[Bau05] Andrej Bauer. Realizability as the connection between computable and constructive mathematics. In *Proceedings of CCA*, 2005.

[BC88] Hans-Juergen Karl Hermann Boehm and Robert Cartwright. *Exact real arithmetic: Formulating real numbers as functions*. Rice University, Department of Computer Science, 1988.

[BCC+06] A. Balluchi, A. Casagrande, P. Collins, A. Ferrari, T. Villa, and A.L. Sangiovanni-Vincentelli. Ariadne: a framework for reachability analysis of hybrid automata. In *Proc. 17th Int. Symp. on Mathematical Theory of Networks and Systems*, Kyoto, 2006.

[BCK+16] Franz Brauße, Pieter Collins, Johannes Kanig, SunYoung Kim, Michal Konečnỳ, Gyesik Lee, Norbert Müller, Eike Neumann, Sewon Park, Norbert Preining, et al. Semantics, logic, and verification of "exact real computation". *arXiv preprint arXiv:1608.05787*, 2016.

[BCRO86] Hans-J Boehm, Robert Cartwright, Mark Riggle, and Michael J O'Donnell. Exact real arithmetic: A case study in higher order programming. In *Proceedings of the 1986 ACM conference on LISP and functional programming*, pages 162–173, 1986.

[BES02a] Andrej Bauer, Martín Hötzel Escardó, and Alex Simpson. Comparing functional paradigms for exact real-number computation. In Peter Widmayer, Stephan Eidenbenz, Francisco Triguero, Rafael Morales, Ricardo Conejo, and Matthew Hennessy, editors, *Automata, Languages and Programming*, pages 488–500, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[Bés02b] Alexis Bés. A survey of arithmetical definability. *Soc. Math. Belgique*, A tribute to Maurice Boffa:1–54, 2002.

[BH98] Vasco Brattka and Peter Hertling. Feasible real random access machines. *Journal of Complexity*, 14(4):490–526, 1998.

[BHW08] Vasco Brattka, Peter Hertling, and Klaus Weihrauch. A tutorial on computable analysis. In *New computational paradigms*, pages 425–491. Springer, 2008.

[Bis67] Errett Bishop. *Foundations of constructive analysis*, volume 60. McGraw-Hill New York, 1967.

[Bra95] Vasco Brattka. Computable selection in analysis. In *Proc. of the Workshop on Computability and Complexity in Analysis, Informatik Berichte, FernUniversit at Hagen*, volume 190, pages 125–138, 1995.

[Bra03] Vasco Brattka. The emperor's new recursiveness: The epigraph of the exponential function in two models of computability. In *Words, Languages & Combinatorics III*, pages 63–72. World Scientific, 2003.

[BSS+89] Lenore Blum, Mike Shub, Steve Smale, et al. On a theory of computation and complexity over the real numbers: *np*-completeness, recursive functions and universal machines. *Bulletin (New Series) of the American Mathematical Society*, 21(1):1–46, 1989.

[BT82a] J.A. Bergstra and J.V. Tucker. Expressiveness and the completeness of hoare's logic. *Journal of Computer and System Sciences*, 25(3):267 – 284, 1982.

[BT82b] Jan A. Bergstra and John V Tucker. Some natural structures which fail to possess a sound and decidable hoare-like logic for their while-programs. *Theoretical Computer Science: the journal of the EATCS*, 17(3):303–315, 1982.

[BV99] Paolo Boldi and Sebastiano Vigna. Equality is a jump. *Theoretical computer science*, 219(1-2):49–64, 1999.

[BVS93] Stephen Brookes and Kathryn Van Stone. Monads and comonads in intensional semantics. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1993.

[BZ98] Christoph Burnikel and Joachim Ziegler. Fast recursive division. 1998.

[CM06] Stefania Corsaro and Marina Marino. Interval linear systems: the state of the art. *Computational Statistics*, 21(2):365–384, 2006.

[CNR11] Pieter Collins, Milad Niqui, and Nathalie Revol. A validated real function calculus. *Mathematics in Computer Science*, 5(4):437–467, 2011.

[Col20] Pieter Collins. Computable random variables and conditioning. *arXiv preprint arXiv:2101.00956*, 2020.

[Coo78] Stephen A Cook. Soundness and completeness of an axiom system for program verification. *SIAM Journal on Computing*, 7(1):70–90, 1978.

[Dei91] Assem Deif. The interval eigenvalue problem. *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, 71(1):61–64, 1991.

[DG93] Pietro Di Gianantonio. A functional approach to computability on real numbers. *Bulletin-European Association For Theoretical Computer Science*, 50:518–518, 1993.

[DG96] Pietro Di Gianantonio. Real number computability and domain theory. *Information and Computation*, 127(1):11–25, 1996.

[Dij75] Edsger W Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18(8):453–457, 1975.

[DJ83] Werner Damm and Bernhard Josko. A sound and relatively complete hoare-logic for a language with higher type procedures. *Acta Informatica*, 20(1):59–101, 1983.

[Dri86] Lou van den Dries. The field of reals with a predicate for the powers of two. *Manuscripta mathematica*, 54:187–196, 1986.

[Eda97] Abbas Edalat. Domains for computation in mathematics, physics and exact real arithmetic. *Bulletin of Symbolic Logic*, 3(4):401–452, 1997.

[EE00] Abbas Edalat and Martın Hötzel Escardó. Integration in real pcf. *Information and Computation*, 160(1-2):128–166, 2000.

[EHS04] Martín Escardó, Martin Hofmann, and Thomas Streicher. On the non-sequential nature of the interval-domain model of real-number computation. *Mathematical Structures in Computer Science*, 14(06):803–814, 2004.

[ES14] Martín Hötzel Escardó and Alex Simpson. Abstract datatypes for real numbers in type theory. In *Rewriting and Typed Lambda Calculi*, pages 208–223. Springer, 2014.

[Esc96] Martín Hötzel Escardó. Pcf extended with real numbers. *Theoretical Computer Science*, 162(1):79–115, 1996.

[Gol91] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.*, 23(1):5–48, March 1991.

[GVL96] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 3rd edition, 1996.

[Her96] Peter Hertling. Topological complexity with continuous operations. *Journal of Complexity*, 12:315–338, 1996.

[Her99] Peter Hertling. A real number structure that is effectively categorical. *Math. Log. Q.*, 45:147–182, 1999.

[Hol94] I. Holand. The loss of the sleipner condeep platform. In Ger M.A. Kusters and Max A.N. Hendriks, editors, *Proc. 1st International DIANA Conference on Computational Mechanics*, pages 25–36, Dordrecht, Netherlands, 1994. Springer Netherlands.

[HVDH20] David Harvey and Joris Van Der Hoeven. Integer multiplication in time O(n log n). *Annals of Mathematics*, 2020.

[JM97] Jean-Marc Jézéquel and Bertrand Meyer. Design by contract: The lessons of ariane. *Computer*, 30(1):129–130, January 1997.

[Kle99] Thomas Kleymann. Hoare logic and auxiliary variables. *Formal Aspects of Computing*, 11(5):541–566, 1999.

[KMP+08] Lutz Kettner, Kurt Mehlhorn, Sylvain Pion, Stefan Schirra, and Chee-Keng Yap. Classroom examples of robustness problems in geometric computations. *Comput. Geom.*, 40(1):61–78, 2008.

[KST18] Akitoshi Kawamura, Florian Steinberg, and Holger Thies. Parameterized complexity for uniform operators on multidimensional analytic functions and ode solving. In *International Workshop on Logic, Language, Information, and Computation*, pages 223–236. Springer, 2018.

[KTD+13] M. Konecny, W. Taha, J. Duracz, A. Duracz, and A. Ames. Enclosing the behavior of a hybrid system up to and beyond a zeno point. In *Proc. 1st IEEE Int. Conf. on Cyber-Physical Systems, Networks, and Applications*, pages 120–125, New York, NY, United States, 2013. Association for Computing Machinery.

[LP20] Donghyun Lim and Sewon Park. Topological aspects on nondetermistic computation. In *한국정보과학회 학술발표논문집*, pages 1107–1108, 2020.

[Luc77] Horst Luckhardt. A fundamental effect in computations on real numbers. *Theoretical Computer Science*, 5(3):321 – 324, 1977.

[LW02] Eugene Loh and G. William Walster. Rump's example revisited. *Reliable Computing*, 8(3):245–248, 2002.

[MKC09] Ramon E Moore, R Baker Kearfott, and Michael J Cloud. *Introduction to interval analysis*, volume 110. Siam, 2009.

[Moo66] Ramon E Moore. *Interval analysis*, volume 4. Prentice-Hall Englewood Cliffs, NJ, 1966.

[Moo14] Ramon E Moore. *Reliability in computing: the role of interval methods in scientific computing*, volume 19. Elsevier, 2014.

[MRE07] J. Raymundo Marcial-Romero and Martín H. Escardó. Semantics of a sequential language for exact real-number computation. *Theoretical Computer Science*, 379(1):120–141, 2007.

[Mül00] Norbert Th Müller. The iRRAM: Exact arithmetic in C++. In *International Workshop on Computability and Complexity in Analysis*, pages 222–252. Springer, 2000.

[Nel89] Greg Nelson. A generalization of dijkstra's calculus. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 11(4):517–561, 1989.

[PC99] Victor Y Pan and Zhao Q Chen. The complexity of the matrix eigenproblem. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 507–516. ACM, 1999.

[PER89] Marian B. Pour-El and J. Ian Richards. *Computability in Analysis and Physics*. Perspectives in Mathematical Logic. Springer, Berlin, 1989.

[Plo76] Gordon D Plotkin. A powerdomain construction. *SIAM Journal on Computing*, 5(3):452–487, 1976.

[Rei79] Karl Reichmann. Abbruch beim Intervall-Gauß-Algorithmus breaking down of the interval gauß algorithm. *Computing*, 22(4):355–361, 1979.

[Rum88] S.M. Rump. Algorithms for verified inclusions: Theory and practice. In R. E. Moore, editor, *Reliability in Computing: The Role of Interval Methods in Scientific Computing*, chapter chapter 1, Computer Arithmetic and Mathematical Software, page 109–126. Academic Press, Boston, 1988.

[Sco70] Dana Scott. *Outline of a mathematical theory of computation*. Oxford University Computing Laboratory, Programming Research Group Oxford, 1970.

[Sha78] Michael Ian Shamos. Computational geometry. *Ph. D. thesis, Yale University*, 1978.

[SS17] Matthias Schröder and Florian Steinberg. Bounded time computation on metric spaces and banach spaces. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12. IEEE, 2017.

[SS18] Svetlana V Selivanova and Victor L Selivanov. Bit complexity of computing solutions for symmetric hyperbolic systems of pdes. In *Conference on Computability in Europe*, pages 376–385. Springer, 2018.

[TKZ18] Holger Thies, Akitoshi Kawamura, and Martin A Ziegler. Average-case polynomial-time computability of hamiltonian dynamics. In *43rd International Symposium on Mathematical Foundations of Computer Science*. Schloss Dagstuhl–Leibniz Center for Informatics, 2018.

[TZ99] John V Tucker and Jeffery I Zucker. Computation by 'while'programs on topological partial algebras. *Theoretical Computer Science*, 219(1-2):379–420, 1999.

[TZ04] John V Tucker and Jeffery I Zucker. Abstract versus concrete computation on metric partial algebras. *ACM Transactions on Computational Logic (TOCL)*, 5(4):611–668, 2004.

[TZ15] JV Tucker and JI Zucker. Generalizing computability theory to abstract algebras. In *Turing's Revolution*, pages 127–160. Springer, 2015.

[vO01] David von Oheimb. Hoare logic for java in isabelle/hol. *Concurrency and Computation: Practice and Experience*, 13(13):1173–1214, 2001.

[VO08] Jaap Van Oosten. *Realizability: an introduction to its categorical side*. Elsevier, 2008.

[Wat82] David S Watkins. Understanding the qr algorithm. *SIAM review*, 24(4):427–440, 1982.

[Wei00] Klaus Weihrauch. Computable analysis: An introduction (texts in theoretical computer science. an EATCS series). 2000.

[Wil68] James Hardy Wilkinson. Global convergene of tridiagonal qr algorithm with origin shifts. *Linear Algebra and its Applications*, 1(3):409–420, 1968.

[YHB07] Nobuko Yoshida, Kohei Honda, and Martin Berger. Logical reasoning for higher-order functions with local state. In *International Conference on Foundations of Software Science and Computational Structures*, pages 361–377. Springer, 2007.

[YSS13] Chee Yap, Michael Sagraloff, and Vikram Sharma. Analytic root clustering: A complete algorithm using soft zero tests. In *Conference on Computability in Europe*, pages 434–444. Springer, 2013.

[ZB04] Martin Ziegler and Vasco Brattka. Computability in linear algebra. *Theoretical Computer Science*, 326(1-3):187–211, 2004.

[Zie05] Martin Ziegler. Computability and continuity on the real arithmetic hierarchy and the power of type-2 nondeterminism. In S. Barry Cooper, Benedikt Löwe, and Leen Torenvliet, editors, *New Computational Paradigms*, pages 562–571, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[Zie12] Martin Ziegler. Real computation with least discrete advice: A complexity theory of nonuniform computability with applications to effective linear algebra. *Annals of Pure and Applied Logic*, 163(8):1108–1139, 2012.

Index

Acknowledgment

(to be written)