Program Extraction from Coinductive Proofs and its Application to Exact Real Arithmetic

Kenji Miyamoto

München 2013

Program Extraction from Coinductive Proofs and its Application to Exact Real Arithmetic

Kenji Miyamoto

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität
München

vorgelegt von
Kenji Miyamoto
aus Osaka, Japan

München, den 18.11.2013

Erstgutachter: Prof. Dr. Helmut Schwichtenberg
Zweitgutachter: Dr. Ulrich Berger
Tag der mündlichen Prüfung: 27.12.2013

Eidesstattliche Versicherung

Hermit erkläre ich an Eidesstatt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

München, 14.11.2013 Kenji Miyamoto

Contents

1 Introduction **1**
- 1.1 Background . . . 1
- 1.2 Inductive and Coinductive Definitions . . . 2
- 1.3 Minlog . . . 3
- 1.4 Contributions . . . 3
- 1.5 Organization of the Thesis . . . 4

2 Theory of Computable Functionals **7**
- 2.1 Partial Continuous Functionals . . . 8
  - 2.1.1 Information Systems and Domains . . . 8
  - 2.1.2 Algebras and Types . . . 10
  - 2.1.3 Partial Continuous Functionals . . . 15
- 2.2 T^+ as an Extension of Gödel's T . . . 17
  - 2.2.1 Constructors and Recursion Operators . . . 18
  - 2.2.2 Destructors and Corecursion Operators . . . 22
  - 2.2.3 Programmable Constants . . . 28
  - 2.2.4 Denotational Semantics . . . 29
- 2.3 Inductive and Coinductive Definitions . . . 32
  - 2.3.1 Inductive Definitions . . . 33
  - 2.3.2 Coinductive Definitions . . . 37
  - 2.3.3 Totality and Cototality . . . 39
  - 2.3.4 Monotonicity . . . 40
  - 2.3.5 Simultaneous Definitions . . . 41
- 2.4 Proofs . . . 44
  - 2.4.1 Natural Deduction . . . 44
  - 2.4.2 Proof Conversion . . . 46
- 2.5 Realizability Interpretation . . . 49
  - 2.5.1 Decorating Inductive and Coinductive Definitions . . . 49
  - 2.5.2 Program Extraction . . . 56
  - 2.5.3 Soundness Theorem . . . 61
- 2.6 Notes . . . 70
  - 2.6.1 T^+ and Partial Continuous Functionals . . . 70
  - 2.6.2 Non-Computational Part of Proofs . . . 70

2.6.3 Inductive and Coinductive Definitions . . . 70
2.6.4 Normalizability . . . 71
2.6.5 Program Extraction in Other Proof Assistants . . . 71

3 Program Extraction in Exact Real Arithmetic 73
3.1 Basics of Exact Real Arithmetic . . . 74
3.1.1 Real Numbers . . . 74
3.1.2 Uniformly Continuous Functions . . . 75
3.2 Rational Number into Signed Digit Stream . . . 76
3.2.1 Definitions . . . 77
3.2.2 Proofs . . . 78
3.2.3 Program Extraction . . . 80
3.2.4 Experiments . . . 80
3.3 Representations of Real Numbers . . . 80
3.3.1 Definitions . . . 81
3.3.2 Proofs . . . 83
3.3.3 Program Extraction . . . 84
3.3.4 Experiments . . . 85
3.4 Average . . . 86
3.4.1 Definitions . . . 88
3.4.2 Proofs . . . 90
3.4.3 Program Extraction . . . 91
3.4.4 Experiments . . . 93
3.5 Representations of Uniformly Continuous Functions . . . 94
3.5.1 Definitions . . . 95
3.5.2 Proofs . . . 97
3.5.3 Program Extraction . . . 100
3.5.4 Experiments . . . 102
3.6 Application of Uniformly Continuous Functions . . . 104
3.6.1 Definitions . . . 104
3.6.2 Proofs . . . 105
3.6.3 Program Extraction . . . 106
3.6.4 Experiments . . . 107
3.7 Composition of Uniformly Continuous Functions . . . 107
3.7.1 Definitions . . . 108
3.7.2 Proofs . . . 108
3.7.3 Program Extraction . . . 110
3.7.4 Experiments . . . 111
3.8 Integration . . . 112
3.8.1 Definitions . . . 112
3.8.2 Proofs . . . 113
3.8.3 Program Extraction . . . 114
3.8.4 Experiments . . . 115

3.9 Notes . . . 115
3.9.1 Exact Real Arithmetic via Streams . . . 115
3.9.2 Program Extraction . . . 116
3.9.3 Abstract Theory . . . 116
3.9.4 Formalizing n-Ary Uniformly Continuous Functions . . . 116

4 Concluding Remarks **119**
4.1 Overview of this Thesis . . . 119
4.2 Contributions . . . 119
4.3 Future Work . . . 120
4.3.1 Proof Normalization . . . 120
4.3.2 Automatic Program Certification . . . 120
4.3.3 Formalizing n-Ary Uniformly Continuous Functions . . . 121

A Proof Assistant Minlog **123**
A.1 Implementation . . . 123
A.1.1 Normalization by Evaluation . . . 123
A.1.1.1 Term Family . . . 124
A.1.1.2 Reflect and Reify . . . 125
A.1.2 Recursion and Corecursion . . . 127
A.1.3 Inductive and Coinductive Definitions . . . 133
A.2 Commentary to `cauchysds.scm` . . . 134
A.2.1 Definitions . . . 135
A.2.2 Proofs . . . 137
A.2.3 Program Extraction . . . 142

Acknowledgments

I would like to express my deepest gratitude to my advisor, Professor Helmut Schwichtenberg. He has given me an opportunity to be a PhD student in the logic group of the university of Munich. I have learnt from him not only mathematical logic but also informatics through my work to develop new features of the proof assistant Minlog. I would like to express my appreciation to Ulrich Berger. He accepted to be the second referee of this Thesis, and also spent a lot of time for me to have scientific discussions whenever we were in the same location. I would like to thank the MALOA project for financial supports[1].

In Munich I had the opportunity to have many colleagues to work together. I would like to thank Fredrik Nordvall Forsberg, Simon Huber, Florian Ranzi, Diana Ratiu and Vesela Yotova for discussions on the Theory of Computable Functionals, Minlog and/or coinduction. I would like to thank Josef Berger, Basil Karadais, Iosif Petrakis, Davide Rinaldi and Pedro Valencia for discussions on constructive mathematics. I would like to thank Professor Masahiko Sato for discussions on formalization of mathematics and proof assistants during his stays in Munich.

I had an opportunity to stay at the Swansea University for ten weeks. I am grateful to Ulrich Berger and Monika Seisenberger for their help during my stay in Swansea. I learnt a lot from scientific discussions with people in the research group of theoretical computer science at the Swansea University. My thank goes to Ulrich Berger, Fredrik Nordvall Forsberg, Tie Hou, Andy Lawrence, Monika Seisenberger and Anton Setzer.

Finally, I wish to thank my family for their support.

[1]The research leading to these results has received funding from the [European Community's] Seventh Framework Programme [FP7/2007-2013] under grant agreement n° 238381.

Abstract

Program extraction has been initiated in the field of constructive mathematics, and it attracts interest not only from mathematicians but also from computer scientists nowadays. From a mathematical viewpoint its aim is to figure out computational meaning of proofs, while from a computer-scientific viewpoint its aim is the study of a method to obtain correct programs. Therefore, it is natural to have both theoretical results and a practical computer system to develop executable programs via program extraction.

In this Thesis we study the computational interpretation of constructive proofs involving inductive and coinductive reasoning. We interpret proofs by translating the computational content of proofs into executable program code. This translation is the procedure we call program extraction and it is given through Kreisel's modified realizability. Here we study a proof-theoretic foundation for program extraction, enriching the proof assistant system Minlog based on this theoretical improvement. Once a proof of a formula is written in Minlog, a program can be extracted from the proof by the system itself, and the extracted program can be executed in Minlog. Moreover, extracted programs are provably correct with respect to the proven formula due to a soundness theorem which we prove. We practice program extraction by elaborating some case studies from exact real arithmetic within our formal theory. Although these case studies have been studied elsewhere, here we offer a formalization of them in Minlog, and also machine-extraction of the corresponding programs.

Zusammenfassung

Die Methode der Programmextraktion hat ihren Ursprung im Bereich der konstruktiven Mathematik, und stößt in letzter Zeit auf viel Interesse nicht nur bei Mathematikern sondern auch bei Informatikern. Vom Standpunkt der Mathematik ist ihr Ziel, aus Beweisen ihre rechnerische Bedeutung abzulesen, während vom Standpunkt der Informatik ihr Ziel die Untersuchung einer Methode ist, beweisbar korrekte Programme zu erhalten. Es ist deshalb naheliegend, neben theoretischen Ergebnissen auch ein praktisches Computersystem zur Verfügung zu haben, mit dessen Hilfe durch Programmextraktion lauffähige Programme entwickelt werden können.

In dieser Doktorarbeit wird eine rechnerische Interpretation konstruktiver Beweise mit induktiven und koinduktiven Definitionen angegeben und untersucht. Die Interpretation geschieht dadurch, daß der rechnerische Gehalt von Beweisen in eine Programmiersprache übersetzt wird. Diese Übersetzung wird Programmextraktion genannt; sie basiert auf Kreisels modifizierter Realisierbarkeit. Wir untersuchen die beweistheoretischen Grundlagen der Programmextraktion und erweitern den Beweisassistenten Minlog auf der Basis der erhaltenen theoretischen Resultate. Wenn eine Formel in Minlog formal bewiesen ist, läßt sich ein Programm aus dem Beweis extrahieren, und dieses extrahierte Programm kann in Minlog ausgeführt werden. Ferner sind extrahierte Programme beweisbar korrekt bezüglich der entsprechenden Formel aufgrund eines Korrektheitsatzes, den wir beweisen werden. Innerhalb unserer formalen Theorie bearbeiten wir einige aus der Literatur bekannte Fallstudien im Bereich der exakten reellen Arithmetik. Wir entwickeln eine vollständige Formalisierung der entsprechenden Beweise und diskutieren die in Minlog automatisch extrahierten Programme.

Publications Included in this Thesis

1. Ulrich Berger, Kenji Miyamoto, Helmut Schwichtenberg, and Monika Seisenberger. Minlog — a tool for program extraction supporting algebras and coalgebras. In Andrea Corradini, Bartek Klin, and Corina Cîrstea, editors, *CALCO*, volume 6859 of *Lecture Notes in Computer Science*, pages 393–399. Springer, 2011 [BMSS11].

2. Kenji Miyamoto and Helmut Schwichtenberg. Program extraction in exact real arithmetic. *Mathematical Structure of Computer Science*, 2013. in press [MS13].

3. Kenji Miyamoto, Fredrik Nordvall Forsberg, and Helmut Schwichtenberg. Program extraction from nested definitions. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *ITP*, volume 7998 of *Lecture Notes in Computer Science*, pages 370–385. Springer, 2013 [MFS13].

The author's contribution to the first paper amounts to 25%. The corresponding contents are coinductive definitions in Section 2.3.2, corecursion operators in Section 2.2.2, program extraction in Section 2.5.2 and the case study in Section 3.2. The author's contribution to the second paper amounts to 50%. The corresponding contents are the case study in Section 3.4. The author's contribution to the third paper amounts to 33%. The corresponding contents are inductive definitions in Section 2.3.1, coinductive definitions in Section 2.3.2, corecursion operators in Section 2.2.2, program extraction in Section 2.5.2 and the case study in Section 3.5 and Section 3.8.

Chapter 1

Introduction

An important aspect of constructive mathematics is that a mathematician has to deal with mathematical objects only by realistic and practicable instructions. As a consequence, proofs in constructive mathematics contain algorithms which compute a solution to the corresponding proposition. Program extraction is a method to synthesize from proofs an executable program which is certified with regarding the proposition as a specification. This Thesis studies program extraction from inductive and coinductive proofs within the Theory of Computable Functionals [SW12], TCF in short. Theoretical results are implemented as additional features of the proof assistant Minlog [Min]. Non-trivial case studies in exact real arithmetic due to Berger and Seisenberger [Ber09, Ber10, Ber11, BS10, BS12] are formalized on a computer as a concrete practice of Minlog.

1.1 Background

The idea of program extraction goes back to the so-called Brouwer-Heyting-Kolmogorov interpretation, BHK interpretation for short, of intuitionistic logic which assigns to each formula a *proof*. Kleene introduced *realizability* [Kle45], a main mathematical tool of this Thesis, which assigns to a formula A another formula stating n realizes A where n is a natural number encoding a *realizer* of A. A realizer is intended to be an evidence which constructively justifies a claim. Kreisel introduced *modified realizability* [Kre59] which adopts Heyting arithmetic, where realizers are given in Gödel's T. Realizability in TCF is based on Kreisel's modified realizability. Within TCF, atomic formulas are given by inductive and coinductive definitions, and realizers are given in T^+, an extension of Gödel's T [Göd58]. One may view realizability as an instance of the BHK interpretation. A procedure called *program extraction* is defined based on realizability which transforms a proof of a formula A into a realizer of A. Considering the formula A as a specification, the realizability statement is read as a correctness statement of a realizer respecting the specification. Program extraction is a way to synthesize a provably correct program with respect to the specification due to a corresponding soundness theorem.

TCF is a framework of program extraction which is based on first-order minimal logic.

The theory is extendable by means of inductive and coinductive definitions. The realizability yields the program extraction procedure in TCF whose target language is T^+. The soundness theorem ensures that a proved formula has a realizer, which is in fact given by program extraction. TCF supports so-called general induction principle and also classical proofs due to the A-translation and Gödel's dialectica interpretation.

1.2 Inductive and Coinductive Definitions

TCF adopts coinduction, considered as a dual of induction, as a main feature of program extraction study. By inductive and coinductive definitions, we add new predicate constants and axioms involving the predicate constants extending TCF. Such predicates are called inductive and coinductive predicates. The main idea is the following. An inductive definition provides a way of construction. As an example, we define even numbers by a predicate Ev which tells us a way to construct even numbers.

$$\mathrm{Ev}\,0, \qquad \forall_n(\mathrm{Ev}\,n \to \mathrm{Ev}(n+2)).$$

Similarly, we can also specify a list of even numbers by a predicate LE.

$$\mathrm{LE}\,[], \qquad \forall_{n,ns}(\mathrm{Ev}\,n \to \mathrm{LE}\,ns \to \mathrm{LE}(n{::}ns)).$$

These instructions of constructions are called *introduction axioms*. Assume P and Q are predicates. We derive an instruction of inspection called an *elimination axiom* which works in a different direction from introduction axioms.

$$\forall_n(\mathrm{Ev}\,n \to P\,0 \to \forall_n(P\,n \to P(n+2)) \to P\,n),$$
$$\forall_{ns}(\mathrm{LE}\,ns \to Q\,[] \to \forall_{n,ns}(\mathrm{Ev}\,n \to Q\,ns \to Q(n{::}ns)) \to Q\,ns).$$

We call P and Q *competitors*. Intuitively, an elimination axiom tells us that the inductively defined predicate is smaller than any competitor if all the introduction axioms replaced by the competitor hold. Obviously it is so-called induction principles in daily mathematics. Coinduction works in the opposite way from induction. A coinductive definition specifies an instruction of inspection called a *closure axiom* instead of one of construction. Taking an example in lists, we can say that a list is the empty list or consists of the head and the tail which are respectively an even number and a list of even numbers. This is formulated by the following formula.

$$\forall_{ns}(\mathrm{LE}\,ns \to ns = [] \vee \exists_{m,ms}(\mathrm{Ev}\,m \wedge \mathrm{LE}\,ms \wedge ns = m{::}ms)).$$

The duality may be clearer if we formulate the introduction axioms of LE in the following logically equivalent one.

$$\forall_{ns}(ns = [] \vee \exists_{m,ms}(\mathrm{Ev}\,m \wedge \mathrm{LE}\,ms \wedge ns = m{::}ms) \to \mathrm{LE}\,ns).$$

Differently from the case of inductive definitions, we derive an instruction of construction called a *greatest-fixed-point axiom*, as follows

$$\forall_{ns}(Q\,ns \to \forall_{ns}(Q\,ns \to ns = [] \vee \exists_{m,ms}(\mathrm{Ev}\,m \wedge Q\,ms \wedge ns = m{::}ms)) \to \mathrm{LE}\,ns).$$

Intuitively, a greatest-fixed-point axiom tells us that a coinductively defined predicate is bigger than any competitor, if the closure axiom replaced by the competitor holds. One can consider a coinductive predicate containing lists whose tails are never inspected to be the empty list, as long as we repeatedly apply the closure axiom. Hence, we can make use of coinduction to reason about possibly infinite lists. This is a meaningful alternative to make use of a function to define a sequence, because an infinite list is represented by coinduction as a simpler ground type object instead of a functional type object. Thanks to it, infinite objects may be observed rather directly in the realm of coinduction. This advantage becomes more evident when we consider a function with an uncountable domain, e.g. function spaces, to define an infinite object.

1.3 Minlog

Since we consider program extraction as a method of program development, it is crucial to have a computer system which is a realization of our study. The Minlog system is a general purpose proof assistant whose competitive feature is program extraction. The development of Minlog was initiated by Schwichtenberg around 1990 as an implementation of TCF [Sch92]. The current version of Minlog supports extraction from classical proofs through the *A-translation* and *Gödel's dialectica interpretation* as well as from constructive proofs involving inductive and coinductive arguments.

We review related proof assistants. The NuPRL system [Nup, BC85] was founded by Bates and Constable. NuPRL is one of the most important realization of the proofs-as-programs paradigm. The PX system by Hayashi [HN87] features q-realizability. For program extraction support of recent competitive other proof assistants as Coq, Isabelle and Agda, see Section 2.6.5.

1.4 Contributions

The contributions of this Thesis are the following:

Theory of program extraction. We study new features to enrich TCF, our theoretical foundation of studying program extraction. In addition to (simultaneous) inductive definitions and (simultaneous) coinductive definitions, we consider nested definitions to combine inductive definitions and coinductive definitions in TCF. This is a reasonable way to use both of induction and coinduction in one definition. Nested definition yields axioms for inductive and coinductive reasoning which also contain nestedness. On the other hand, our formal calculus T^+ is enriched to accommodate computational aspects of nestedness.

The program extraction procedure is given through a soundness theorem, according to which, if a formula A is proven, then there exists a realizer t such that t realizes A. In TCF the target language of program extraction is T^+. We extend the soundness theorem to accommodate (strong) induction and (strong) coinduction with nestedness.

Implementation. The theoretical improvements are implemented as additional features of the proof assistant Minlog. Our main contribution is the implementation of nested algebras and nested recursion and corecursion operators as well as of nested inductive and coinductive definitions. The program extraction procedure is also enriched to accommodate the nestedness. The term normalization of arbitrary recursion operators is particularly done via *Normalization by Evaluation* [BS91, BES03], which is an effective method to implement normalization based on denotational semantics. In order to handle nested recursion operators, the Normalization by Evaluation of Minlog is improved. For corecursion operators Minlog offers bounded unfolding instead.

Practicing constructive mathematics. We apply TCF and Minlog to concrete case studies in exact real arithmetic involving real numbers and uniformly continuous functions. In order to study them, we exploit two kinds of representations: one is the function type representation and the other is the ground type coinductive representation. For simplicity, we restrict our real numbers to be in the interval $[-1, 1]$. The function type representation of real numbers, say the type-1 representation, is a pair of a Cauchy sequence and a Cauchy modulus which are of function types. The ground type coinductive representation of real numbers, say the type-0 representation, is a (possibly) infinite list of digits. The more number of digits we observe from a list, the more precise approximation of the represented real number becomes. The representations of uniformly continuous functions follow the same pattern. We understand real numbers and uniformly continuous functions as objects which are approximated as good as we require and are observable more directly than functions. Within coinduction, we achieve such representations straightforwardly; this is why we find an advantage in studying exact real arithmetic coinductively. We present formalization of case studies by Berger and Seisenberger in exact real arithmetic. We extract from proofs non-trivial programs dealing with real numbers and uniformly continuous functions in Minlog system.

1.5 Organization of the Thesis

This Thesis is organized in the following way. Chapter 2 describes TCF, our theoretical foundation of program extraction. We review the domain-theoretic background of partial continuous functionals. Free algebras are taken as ground types of our formal calculus T^+. We study nested algebras as well as simultaneous algebras. We use T^+ as the target language of our program extraction as well as the terms of our logical language of minimal logic. We study inductive and coinductive definitions in first-order minimal logic, and the notion of proofs and the realizability are given. Based on the realizability we define program

extraction which transforms proofs into a program code in T^+. The soundness theorem, which states that the proven formula has a realizer, is proven. We practice TCF in an elementary part of exact real arithmetic. We briefly review real numbers and uniformly continuous functions in exact real arithmetic, and give case studies in exact real arithmetic due to Berger and Seisenberger in Chapter 3. These case studies involving coinduction are formalized in the proof assistant Minlog as well as in the Thesis. The downloadable Minlog package contains the proof scripts corresponding to Chapter 3 which offer machine-extracted programs running on a computer. Appendix A consists of two parts. The first part describes the implementation of Minlog focusing on Normalization by Evaluation, recursion and corecursion operators in T^+ and inductive and coinductive definitions. The second part explains a Minlog proof script from Section 3.3 in detail.

Chapter 2

Theory of Computable Functionals

In this chapter we study the theoretical foundation for program extraction called *Theory of Computable Functionals*, TCF in short. TCF provides our theoretical basis to practice program extraction on the computer as well as on the paper. The remarkable character of TCF is its concreteness, which becomes an advantage if one desires to have an implementation of such a theory on a computer.

Computable objects in TCF are *partial continuous functionals* which model our notion of higher type abstract computability specified by the following two principles: the *finite support principle* and the *monotonicity principle*. During the evaluation of some functional Φ, there are only finitely many inputs $\varphi_0, \ldots, \varphi_{n-1}$ used. Moreover, each of φ_i must be presented to Φ in a finite form. This is the finite support principle. Assume $\Phi(\varphi_0)$ evaluates to a value k and let φ_1 be more informative than φ_0. Then the $\Phi(\varphi_1)$ results in k as well. This is the monotonicity principle. Then, the notion of abstract computability is formulated as follows: an object is computable if its set of finite approximation is primitive recursively enumerable. Such functionals are represented by the term language T^+ which is an extension of Gödel's T.

The logic of TCF is first order minimal logic formulated in natural deduction. The logic can be extended by inductive and coinductive definitions. Nested definitions and simultaneous definitions enrich the expressivity of definitions in TCF. Especially nested definitions allow a combination of inductive definitions and coinductive ones. Classical logic is also available as a fragment of minimal logic. We can extract from proofs computational content via program extraction. Program extraction is based on Kreisel's modified realizability interpretation which enjoys the soundness theorem in TCF. Realizability statements, e.g. a term t realizes a formula A, are formulas in TCF, and its proof is also given in TCF.

This chapter is organized as follows. In Section 2.1, we provide a concrete formulation of partial continuous functionals due to Schwichtenberg [SW12] by means of Scott-Ershov domains. We briefly review that our computational objects are represented by *ideals* which are special elements in domains. In Section 2.2, we study T^+, an extension of Gödel's T. This is a typed lambda calculus with constants as recursion and corecursion operators. We briefly also review that the denotational semantics of T^+ is given via ideals. A term in Gödel's T with arbitrary algebras denotes a *total* ideal. A corecursion operator, a proper

extension from Gödel's T, denotes a *cototal* ideal of a base type which can represent a non-well founded object. In Section 2.3, we describe inductive and coinductive definitions in TCF. Nested inductive/coinductive definitions and simultaneous definitions are also given. Nested definitions specify a new predicate by using old ones, and simultaneous definitions specify more than one predicate at the same time. In Section 2.4, we describe natural deduction and proof normalization involving inductive and coinductive definitions. The notion of proof terms is also introduced based on the Curry-Howard correspondence. In Section 2.5, we describe Kreisel's modified realizability interpretation. The logical connectives of TCF are decorated in order to study computational meaning of them with greater details, then we introduce the notion of realizability and program extraction. We also give the soundness theorem, which claims that the proved proposition is realized by the extracted program from its proof. This justifies the correctness of the program extraction.

2.1 Partial Continuous Functionals

We study higher type computability via partial continuous functionals. In this section, we describe the notion of *ideals* which indeed are concrete representations of partial continuous functionals. In order to provide a concrete construction of such objects we make use of Scott's information systems [Sco82, Win93], then ideals come as special sets which are *consistent* and *deductively closed*. Domain theoretically speaking, an ideal is an element in Scott-Ershov domains with free algebras as ground types. In Section 2.1.1 we study Scott's information systems as a way of constructing domains. In Section 2.1.2 we study types whose ground types are given by (free) algebras. In Section 2.1.3 we study concrete information systems based on types from which we define the notion of a partial continuous functional of finite type.

2.1.1 Information Systems and Domains

An information system consists of three ingredients: the *tokens*, the *consistent sets* and the *entailment relation*. Information systems can be viewed as deduction systems. Tokens are assertions about a computation. A consistent set contains tokens which can be true assertions at the same time. The entailment is a way to deduce from a set of assertions another assertion. We first give an account on information systems in general, then we consider concrete ones which are defined on types.

Definition 2.1.1 (Information systems)**.** Let U, V range over finite sets. An information system is a structure $\langle \mathrm{Tok}, \mathrm{Con}, \vdash \rangle$ such that

$$
\begin{aligned}
&U \subseteq V \in \mathrm{Con} \to U \in \mathrm{Con},\\
&\{a\} \in \mathrm{Con} \text{ for any } a \in \mathrm{Tok},\\
&U \vdash a \to U \cup \{a\} \in \mathrm{Con},\\
&a \in U \in \mathrm{Con} \to U \vdash a,\\
&U, V \in \mathrm{Con} \to \forall_{a \in V}(U \vdash a) \to V \vdash b \to U \vdash b.
\end{aligned}
$$

Intuitively, tokens carry informative content of computation which should be consistent and deductively closed. Assume one computation means 7, then it should *not* mean 13 at the same time, and it should remain to be 7 in the future. We formulate such tokens as ideals.

Definition 2.1.2 (Ideals)**.** Let IS be an information system $\langle \mathrm{Tok}, \mathrm{Con}, \vdash \rangle$. The ideals $|\mathrm{IS}|$ are subsets of Tok whose elements $x \in |\mathrm{IS}|$ are

1. consistent: $a \subseteq x \to a \in \mathrm{Con}$,
2. deductively closed: $x \supseteq U \vdash a \to a \in x$.

A cpo $(|\mathrm{IS}|, \subseteq, \emptyset)$ forms a *Scott-Ershov domain*, namely, a bounded complete algebraic cpo. In our theory of computability functionals are of higher type in general. We consider function spaces between information systems.

Definition 2.1.3 (Function spaces)**.** Suppose that IS_A and IS_B are $\langle \mathrm{Tok}_A, \mathrm{Con}_A, \vdash_A \rangle$ and $\langle \mathrm{Tok}_B, \mathrm{Con}_B, \vdash_B \rangle$, respectively, and I be a finite set of indices. We define $\mathrm{IS}_A \to \mathrm{IS}_B$ to be $\langle \mathrm{Tok}, \mathrm{Con}, \vdash \rangle$ such that

$$\begin{aligned}
&\mathrm{Tok} \coloneqq \mathrm{Con}_A \times \mathrm{Tok}_B,\\
&\{(U_i, b_i) \mid i \in I\} \in \mathrm{Con} \coloneqq \forall_{J \subseteq I}(\textstyle\bigcup_{j \in J} U_j \in \mathrm{Con}_A \to \{b_j \mid j \in J\} \in \mathrm{Con}_B),\\
&W \vdash (U, b) \coloneqq WU \vdash_B b,
\end{aligned}$$

where for $W \coloneqq \{(U_i, b_i) \mid i \in I\}$ and $U \in \mathrm{Con}_A$ an *application* WU is defined to be $\{(U_i, b_i) \mid i \in I\}U \coloneqq \{b_i \mid U \vdash_A U_i\}$.

We introduce *approximable maps* as an alternative characterization of the function spaces. An approximable map r is a relation between Con_A and Tok_B which represents an information respecting map from IS_A into IS_B. Intuitively $r(U, b)$ means that if we are given the information $U \in \mathrm{Con}_A$, then we find at least the token $b \in \mathrm{Tok}_B$ in the value.

Definition 2.1.4 (Approximable maps)**.** Suppose that IS_A and IS_B are $\langle \mathrm{Tok}_A, \mathrm{Con}_A, \vdash_A \rangle$ and $\langle \mathrm{Tok}_B, \mathrm{Con}_B, \vdash_B \rangle$, respectively. A relation $r \subseteq \mathrm{Con}_A \times \mathrm{Tok}_B$ is an approximable map if the following conditions are satisfied.

1. If $r(U, b_0), \ldots, r(U, b_{n-1})$, then $\{b_0, \ldots, b_{n-1}\} \in \mathrm{Con}_B$.
2. If $r(U, b_0), \ldots, r(U, b_{n-1})$ and $\{b_0, \ldots, b_{n-1}\} \vdash_B b$, then $r(U, b)$.
3. If $r(U', b)$ and $U \vdash_A U'$, then $r(U, b)$.

By the following proposition ideals in a function space are indeed approximable maps [SW12].

Proposition 2.1.5 (Ideals in a function space are approximable maps)**.** *Let* IS_A *and* IS_B *be information systems. The ideals of* $\mathrm{IS}_A \to \mathrm{IS}_B$*, namely,* $|\mathrm{IS}_A \to \mathrm{IS}_B|$ *are exactly the approximable maps from* IS_A *to* IS_B*.*

2.1.2 Algebras and Types

We introduce *types* for our theory of computability. The ground types are given by *algebras*. An algebra is specified by constructor types, and provides *constructor symbols* of the constructor types. For each algebra ι, there is at least one object of type ι finitely built from the constructor symbols. Such an object is an *inhabitant* and ι is said to be *inhabited*. We define type forms as follows.

Definition 2.1.6 (Type forms)**.** Let α and ξ be variables and k be positive. Type forms are inductively defined as follows.

$$\rho, \sigma := \alpha \mid \rho \to \sigma \mid \mu_\xi((\rho_{i\nu})_{\nu<n_i} \to \xi)_{i<k}.$$

In an expression $(\rho_{i\nu})_{\nu<n_i} \to \xi$, $\rho_{i\nu}$ is a parameter argument if $\xi \notin \mathrm{FV}(\rho_{i\nu})$, $\rho_{i\nu}$ is a recursive argument if $\xi \in \mathrm{FV}(\rho_{i\nu})$ and $\rho_{i\nu}$ is a nested argument if ξ appears as an argument in another expression of the form $\mu_\zeta((\rho'_{i\nu})_{\nu<n'_i} \to \zeta)_{i<k'}$ in $\rho_{i\nu}$.

Note that $\rho_0 \to \rho_1 \to \ldots \to \rho_{n-1} \to \sigma$ is abbreviated as $(\rho_\nu)_{\nu<n} \to \sigma$. We define $\mathrm{FV}(\rho)$ to be the set of free type variables in ρ.

Definition 2.1.7 (Free variables in a type form)**.** Define free variables in a type form τ by induction.

$$\begin{aligned}
&\mathrm{FV}(\alpha) := \{\alpha\},\\
&\mathrm{FV}(\rho \to \sigma) := \mathrm{FV}(\rho) \cup \mathrm{FV}(\sigma),\\
&\mathrm{FV}(\mu_\xi((\rho_{i\nu})_{\nu<n_i} \to \xi)_{i<k}) := \cup_{i<k}\mathrm{FV}((\rho_{i\nu})_{\nu<n_i} \to \xi) \setminus \{\xi\}.
\end{aligned}$$

We say $\mathrm{FV}(\tau)$ to be free variables of τ.

We denote $\rho(\vec{\alpha})$ to express that $\rho(\vec{\sigma})$ means replacing $\vec{\alpha}$ in ρ by $\vec{\sigma}$. We assume standard α-equivalence and substitution. The result of substituting $\vec{\sigma}$ for $\vec{\alpha}$ in $\rho(\vec{\alpha})$ is denoted by $\rho(\vec{\sigma})$. We define $\mathrm{SP}_{\vec{\zeta}}(\rho)$ by induction on ρ. When $\mathrm{SP}_{\vec{\zeta}}(\rho)$, the occurrences of $\vec{\zeta}$ in ρ are strictly positive, namely, not in the left hand side of an arrow type. Assume $\mathrm{SP}_{\vec{\zeta}}(\mu_\xi((\rho_{i\nu})_{\nu<n_i} \to \xi)_{i<k})$, then it is also ensured that each argument $\rho_{i\nu}$ of a constructor type does not have the type variable ξ in the left hand side of an arrow type.

Definition 2.1.8 (Strictly positive occurrence of type)**.** Let ρ and σ range over type forms. We define $\mathrm{SP}_{\vec{\zeta}}(\rho)$ by induction on ρ.

$$\mathrm{SP}_{\vec{\zeta}}(\beta), \quad \frac{\mathrm{FV}(\rho) \cap \vec{\zeta} = \emptyset \quad \mathrm{SP}_{\vec{\zeta}}(\sigma)}{\mathrm{SP}_{\vec{\zeta}}(\rho \to \sigma)}, \quad \frac{\text{For all } i<k,\ \nu<n_i,\ \mathrm{SP}_{\vec{\zeta},\xi}(\rho_{i\nu})}{\mathrm{SP}_{\vec{\zeta}}(\mu_\xi((\rho_{i\nu})_{\nu<n_i} \to \xi)_{i<k})}.$$

When $\mathrm{SP}_{\vec{\zeta}}(\rho)$, we say that the occurrences of $\vec{\zeta}$ in ρ are strictly positive. When $\vec{\zeta}$ is empty, we write $\mathrm{SP}(\rho)$ instead of $\mathrm{SP}_{\vec{\zeta}}(\rho)$.

We define inhabitedness of a type form. A type is *inhabited* if there exists an inhabitant of the type assuming the inhabitedness of its type parameters. A type is *absolutely inhabited* if its inhabitedness is *not* dependent on the inhabitednesses of its type parameters.

Definition 2.1.9 (Inhabitedness)**.** We inductively define Inhab.

$$\frac{\alpha \notin \vec{\zeta}}{\mathrm{Inhab}_{\vec{\zeta}}(\alpha)}\ ,\quad \frac{\mathrm{Inhab}_{\vec{\zeta}}(\sigma)}{\mathrm{Inhab}_{\vec{\zeta}}(\rho \to \sigma)}\ ,\quad \frac{\text{There exists } i, \text{ for all } \nu < n_i,\ \mathrm{Inhab}_{\vec{\zeta},\xi}(\rho_{i\nu})}{\mathrm{Inhab}_{\vec{\zeta}}(\mu_\xi((\rho_{i\nu})_{\nu<n_i} \to \xi)_{i<k})}\ .$$

If $\vec{\zeta}$ is empty, we omit the subscript and denote $\mathrm{Inhab}(\rho)$. When $\mathrm{Inhab}(\rho)$ is derived, ρ is inhabited. When $\mathrm{Inhab}_{\mathrm{FV}(\rho)}(\rho)$ is derived, ρ is absolutely inhabited. A derivation of $\mathrm{Inhab}_{\vec{\zeta}}(\rho)$ is *canonical* if in each application of the third rule the smallest possible index i is chosen.

In the above definition, the subscripts $\vec{\xi}$ in $\mathrm{Inhab}_{\vec{\xi}}(\rho)$ are type variables whose inhabitedness cannot be used to claim the inhabitedness of ρ. Types are inductively defined in the following way.

Definition 2.1.10 (Types)**.** We inductively define Ty.

$$\mathrm{Ty}(\alpha),\quad \frac{\mathrm{Ty}(\rho)\quad \mathrm{Ty}(\sigma)}{\mathrm{Ty}(\rho \to \sigma)}\ ,$$
$$\frac{\text{For all } i<k,\ \nu<n_i,\ \mathrm{Ty}(\rho_{i\nu}) \text{ and } \mathrm{SP}_\xi(\rho_{i\nu})\quad \mathrm{Inhab}(\mu_\xi((\rho_{i\nu})_{\nu<n_i} \to \xi)_{i<k})}{\mathrm{Ty}(\mu_\xi((\rho_{i\nu})_{\nu<n_i} \to \xi)_{i<k})}\ .$$

When $\mathrm{Ty}(\rho)$ holds, ρ is a type.

In the last rule, the left premise is to ensure algebra definitions to be from strictly positive type forms and the right premise is to ensure the inhabitedness. Algebras come as special kinds of type.

Definition 2.1.11 (Algebra)**.** We call $\iota \coloneqq \mu_\xi((\rho_{i\nu})_{\nu<n_i} \to \xi)_{i<k}$ an algebra if ι is a type.

Assume $\iota \coloneqq \mu_\xi((\rho_{i\nu})_{\nu<n_i} \to \xi)_{i<k}$. When $\mathrm{Ty}(\iota)$, we refer to each $(\rho_{i\nu})_{\nu<n_i} \to \xi$ for each i a *constructor type* of ι. Conventionally we use ι as a variable ranging over algebras and κ over constructor types. An algebra provides for each constructor type a *constructor symbol* $\mathrm{C}_{\iota,i}$. We abbreviate $\mathrm{C}_{\iota,i}$ as C_i if there is no confusion. Algebras are always inhabited, hence there is no empty algebra.

Example 2.1.12 (Inhabitedness of algebras)**.** We give examples and non-examples of inhabitedness of algebras. Define $\mathbf{L}_\alpha$ to be $\mu_\xi(\xi, \alpha \to \xi \to \xi)$ and $\mathbf{L}^+_\alpha$ to be $\mu_\xi(\alpha \to \xi, \alpha \to \xi \to \xi)$. Then, $\mathbf{L}_\alpha$ and $\mathbf{L}^+_{\mathbf{L}(\alpha)}$ are absolutely inhabited, and $\mathbf{L}^+_\alpha$ is inhabited. Define $\alpha + \beta$ to be $\mu_\xi(\alpha \to \xi, \beta \to \xi)$ and ι_α to be $\mu_\xi(\alpha + \xi \to \xi)$. A type form $\mu_\xi(\iota_\xi \to \xi)$ is *not* inhabited, hence not a type. Define $\alpha \times \beta$ to be $\mu_\xi(\alpha \to \beta \to \xi)$. A type form $\mu_\xi(\alpha \times \xi \to \xi)$ is *not* inhabited, hence not a type. Both of $\mu_\xi(\xi \to \xi)$ and $\mu_\xi(\xi, (\xi \to \xi) \to \xi)$ are not inhabited.

We take a look at examples of algebras. Here, conventional names of the constructors are also given. We follow this convention unless it is ambiguous. For convenience, we can write C^κ instead of κ in order to specify also the constructor's name C.

Example 2.1.13 (Algebras)**.** We first take a look at simple algebra definitions.

$$\mathbf{U} \coloneqq \mu_\xi(\mathsf{U}^\xi) \qquad \text{(Unit)}$$
$$\mathbf{B} \coloneqq \mu_\xi(\mathsf{T}^\xi, \mathsf{F}^\xi) \qquad \text{(Booleans)}$$
$$\mathbf{SD} \coloneqq \mu_\xi(\mathsf{L}^\xi, \mathsf{M}^\xi, \mathsf{R}^\xi) \qquad \text{(Signed digits)}$$
$$\mathbf{N} \coloneqq \mu_\xi(0^\xi, \mathsf{S}^{\xi\to\xi}) \qquad \text{(Natural numbers)}$$
$$\mathbf{P} \coloneqq \mu_\xi(1^\xi, \mathsf{S}_0^{\xi\to\xi}, \mathsf{S}_1^{\xi\to\xi}) \qquad \text{(Positive numbers)}$$
$$\mathbf{Z} \coloneqq \mu_\xi(0^\xi, \mathsf{P}^{\mathbf{P}\to\xi}, \mathsf{N}^{\mathbf{P}\to\xi}) \qquad \text{(Integers)}$$
$$\mathbf{Q} \coloneqq \mu_\xi(\#^{\mathbf{Z}\to\mathbf{P}\to\xi}) \qquad \text{(Rational numbers)}$$
$$\mathbf{D} \coloneqq \mu_\xi(0^\xi, \mathsf{D}^{\xi\to\xi\to\xi}) \qquad \text{(Derivations)}$$

The unit algebra is a singleton. The boolean algebra consists of two constructors which are interpreted as true and false. We use the signed digits in particular to study exact real arithmetic. The natural number algebra represents natural numbers by 0 and the successor. The positive number algebra represents binary numbers. Informally, $\mathsf{S}_{b_0}(\ldots(\mathsf{S}_{b_{n-1}}\,1)\ldots)$ means a binary number $1b_{n-1}\ldots b_0$, i.e., a number $\sum_{i=0}^{n-1} 2^i b_i + 2^n$. From a positive number, the positive and the negative part of integers are constructed by injection. The zero element is separately given by the other constructor. In the rational number algebra, a rational number is a pair of a positive number and an integer. The derivation algebra represents binary trees which are constructed by the leaf and the node.

Example 2.1.14 (Algebras with parameters)**.** We give algebra definitions with parameter types.

$$\mathbf{L}_\alpha \coloneqq \mu_\xi([]^\xi, ::^{\alpha\to\xi\to\xi}) \qquad \text{(Lists)}$$
$$\times_{\alpha,\beta} \coloneqq \mu_\xi(\mathsf{Pair}^{\alpha\to\beta\to\xi}) \qquad \text{(Product)}$$
$$+_{\alpha,\beta} \coloneqq \mu_\xi(\mathsf{InL}^{\alpha\to\xi}, \mathsf{InR}^{\beta\to\xi}) \qquad \text{(Sum)}$$
$$\mathbf{Uysum}_\alpha \coloneqq \mu_\xi(\xi, \alpha\to\xi) \qquad \text{(Unit sum)}$$

Type parameters can be written with parentheses, e.g., $\mathbf{L}(\alpha)$, instead of subscripts. We accept the infix notation for $\times_{\alpha,\beta}$ and $+_{\alpha,\beta}$, e.g., $\alpha\times\beta$ and $\alpha+\beta$, respectively.

The list algebra represents lists of objects of an arbitrary type α. The product algebra makes a pair, e.g. a pair of x^α and y^β written as $\mathsf{Pair}xy$ which is abbreviated as $\langle x, y\rangle$. The sum algebra is for the left and the right injection. The unit sum is so-called *maybe algebra*, which is either nothing or just an object of α for example in Haskell. We may call its constructors None and Just. For them, we also write $\mathsf{InL}\,\mathsf{U}$ and InR, respectively, by abusing constructors of $\mathbf{U}$ and $+$.

Example 2.1.15 (Nested algebras)**.** Using previously defined parameterized algebras, algebras can be nested.

$$\begin{aligned}
&\mathbf{Nodd} \coloneqq \mu_\xi(\mathsf{S}^{\mathbf{Uysum}_\xi \to \xi}) && \text{(Nested odd numbers)}\\
&\mathbf{Nt} \coloneqq \mu_\xi(\mathsf{Br}^{\mathbf{L}_\xi \to \xi}) && \text{(Nested trees)}\\
&\mathbf{R}_\alpha \coloneqq \mu_\xi(\mathsf{Put}^{\mathbf{SD} \to \alpha \to \xi}, \mathsf{Get}^{\xi \to \xi \to \xi \to \xi}),\\
&\mathbf{W} \coloneqq \mu_\xi(\mathsf{Stop}^{\xi}, \mathsf{Cont}^{\mathbf{R}_\xi \to \xi}) && \text{(Read-write machines)}
\end{aligned}$$

The nested odd number algebra represents even and odd numbers. Consider None of type $\mathbf{Uysum}_{\mathbf{Nodd}}$ to be zero, an even. Then, $\mathsf{S}\,\mathsf{None}$ is one, an odd, $\mathsf{Just}(\mathsf{S}\,\mathsf{None})$ is two, an even, $\mathsf{S}(\mathsf{Just}(\mathsf{S}\,\mathsf{None}))$ is three, an odd, and so on. The nested tree algebra represents trees of arbitrary height with arbitrary branching. The argument of Br is a list of $\mathbf{Nt}$, which corresponds to each branch from this node. The read-write machine algebra is a kind of a branching tree which plays a special role in exact real arithmetic. We study it in detail in Section 3.5.

Example 2.1.16 (Non-finitary algebras)**.** We give non-finitary algebras. Constructor types can be of higher types.

$$\begin{aligned}
&\mathbf{O} \coloneqq \mu_\xi(\mathsf{0}^{\xi}, \mathsf{S}^{\xi \to \xi}, \mathsf{Sup}^{(\mathbf{N} \to \xi) \to \xi}) && \text{(Ordinals)}\\
&\mathbf{T}_0 \coloneqq \mathbf{N}, \quad \mathbf{T}_{n+1} \coloneqq \mu_\xi(\xi, (\mathbf{T}_n \to \xi) \to \xi) && \text{(Trees)}
\end{aligned}$$

Let $\vec{\xi}$ be a list of types of length n and $i < n$. We define $\vec{\xi}^{(i)}$, the i-th tail of $\vec{\xi}$, as follows: $\vec{\xi}^{(0)} \coloneqq \vec{\xi}$, $(\zeta, \vec{\xi})^{(j+1)} \coloneqq \vec{\xi}^{(j)}$. We modify the above definitions to support *simultaneous algebras*.

Definition 2.1.17 (Simultaneous algebras)**.** We modify type forms as follows. Let l be the length of $\vec{\xi}$, a non-empty list of type variables.

$$\rho, \sigma \coloneqq \alpha \mid \rho \to \sigma \mid \mu_{\vec{\xi}}((\rho_{i\nu})_{\nu<n_i} \to \xi_{j_i})_{i<k},$$

where j is a surjection from $i \in \{0, 1, \ldots, k-1\}$ to $j_i \in \{0, 1, \ldots, l-1\}$. We also replace the definitions involving $\mu_\xi((\rho_{i\nu})_{\nu<n_i} \to \xi)_{i<k}$ by the following.

$$\begin{aligned}
&\mathrm{FV}(\mu_{\vec{\xi}}((\rho_{i\nu})_{\nu<n_i} \to \xi_{j_i})_{i<k}) \coloneqq \cup_{i<k}\mathrm{FV}((\rho_{i\nu})_{\nu<n_i} \to \xi_{j_i}) \setminus \{\vec{\xi}\},\\
&\frac{\text{for all } m < |\vec{\xi}|, \text{ there exists } i,\ j_i = m \text{ and for all } \nu < n_i,\ \mathrm{Inhab}_{\vec{\zeta},\vec{\xi}^{(m)}}(\rho_{i\nu})}{\mathrm{Inhab}_{\vec{\zeta}}(\mu_{\vec{\xi}}((\rho_{i\nu})_{\nu<n_i} \to \xi_{j_i})_{i<k})},\\
&\frac{\text{for all } i < k,\ \nu < n_i,\ \mathrm{Ty}(\rho_{i\nu}) \text{ and } \mathrm{SP}_{\vec{\xi}}(\rho_{i\nu}) \quad \mathrm{Inhab}(\mu_{\vec{\xi}}((\rho_{i\nu})_{\nu<n_i} \to \xi_{j_i})_{i<k})}{\mathrm{Ty}(\mu_{\vec{\xi}}((\rho_{i\nu})_{\nu<n_i} \to \xi_{j_i})_{i<k})}.
\end{aligned}$$

We call $\vec{\iota} := \mu_{\vec{\xi}}((\rho_{i\nu})_{\nu<n_i} \to \xi_{j_i})_{i<k}$ simultaneous algebras if $\mathrm{Ty}(\vec{\iota})$.

The second rule in the above definition is modified to check the inhabitedness of each algebra. We take a look at examples of algebras.

Example 2.1.18 (Simultaneous algebras)**.**

$$\mathbf{Even}, \mathbf{Odd} \coloneqq \mu_{\xi,\zeta}(\xi, \zeta \to \xi, \xi \to \zeta) \qquad \text{(Even and odd numbers)}$$
$$\mathbf{Ts}, \mathbf{T} \coloneqq \mu_{\xi,\zeta}(\xi, \zeta \to \xi \to \xi, \xi \to \zeta) \qquad \text{(Tree lists and trees)}$$

Tree lists and trees are also called forests and trees [Pau00, Coq13]. We can observe the similarity between simultaneous algebras and nested ones. The algebras $(\mathbf{Even}, \mathbf{Odd})$ can work in a similar way as the pair of **Uysum** and **Nodd**. Also algebras $(\mathbf{Ts}, \mathbf{T})$ can work in a similar way as the pair of $\mathbf{L_{Nt}}$ and **Nt**. The simultaneity and the nestedness can be mixed.

Example 2.1.19 (Simultaneous nested algebra)**.**

$$\mathbf{Not}, \mathbf{Net} \coloneqq \mu_{\zeta,\xi}(\mathbf{L}_\zeta \to \xi, \mathbf{L}_\xi \to \zeta) \qquad \text{(Nested odd/even trees)}$$

We give translations between simultaneous algebras and nested algebras. Let $\vec{\iota}$ be simultaneous algebras $\mu_{\vec{\xi}}(\vec{\kappa})$ of length l. Suppose that for a given $m < l$, $\mathrm{Inhab}_{(\xi_i)_{i<m}}(\vec{\kappa}')$ holds, where $\vec{\kappa}'$ is the greatest sublist of $\vec{\kappa}$ such that the value types of $\vec{\kappa}'$ is ξ_m. We can define $\iota'_m(\vec{\alpha})$ instead of ι_m where all the depended simultaneous algebras in ι_m are replaced by parameter types, and also $\vec{\iota}'$ of length $l-1$ which is similar to $\vec{\iota}$ but ι_m is dropped and is replaced by the independently defined $\iota'_m(\vec{\alpha})$.

Definition 2.1.20 (Translation from simultaneity to nestedness)**.** Assume simultaneous algebras $\vec{\iota} = \mu_{\vec{\xi}}(\vec{\kappa})$ of length l are given. Suppose that for a given $m < l$, $\mathrm{Inhab}_{(\xi_i)_{i<m}}(\kappa_i)_{k_{m-1} \le i < k_m}$ holds, where $k_0 = 0$, $k_{l-1} = |\vec{\kappa}|$, $k_i < k_{i+1}$ and all constructor types with the value type ξ_m are listed by indices i, $k_{m-1} \le i < k_m$. We define $\mathrm{nest}_m(\vec{\iota})$ to be the pair of algebras $\mu_{\xi_m}(\kappa_i)_{k_{m-1} \le i < k_m}$ and $\mu_{\vec{\xi} - \xi_m}(\kappa'_0, \ldots, \kappa'_{k_{m-1}-1}, \kappa'_{k_m}, \ldots, \kappa'_{|\vec{\kappa}|})$, where $\vec{\xi} - \xi_m$ is the sublist of $\vec{\xi}$ excluding ξ_m, and κ'_i is obtained by substituting each occurrence of ξ_m by $\mu_{\xi_m}(\kappa_i)_{k_{m-1} \le i < k_m}$.

Definition 2.1.21 (Translation from nestedness to simultaneity)**.** Assume an algebra $\iota(\alpha) = \mu_\xi(\vec{\kappa}_0)$ with at least one parameter type α and (simultaneous) algebras $\vec{\iota} = \mu_{\vec{\xi}}(\vec{\kappa}_1)$ of length l are given. Suppose that for some $m < l$, $\iota(\xi_m)$ appears in $\vec{\kappa_1}$. We define $\mathrm{simul}(\iota, \vec{\iota})$ to be algebras $\mu_{\zeta,\vec{\xi}}(\vec{\kappa}'_0, \vec{\kappa}'_1)$, where $\vec{\kappa}'_0$ is obtained by replacing α in $\vec{\kappa}_0$ by ξ_m and $\vec{\kappa}'_1$ is obtained by replacing $\iota(\xi_m)$ in $\vec{\kappa_1}$ by ζ.

For simplicity, we do not consider to single out simultaneous parameterized algebras from simultaneous algebras, since it is always possible to single out non-simultaneous parameterized algebra. We take a look at an example.

Example 2.1.22 ($(\mathbf{Ts}, \mathbf{T})$ and **Nt**)**.** Consider $(\mathbf{Ts}, \mathbf{T}) = \mu_{\xi_0,\xi_1}(\xi_0, \xi_1 \to \xi_0 \to \xi_0, \xi_0 \to \xi_1)$, then $\mathrm{Inhab}(\xi_0, \xi_1 \to \xi_0 \to \xi_0)$ holds. We can introduce nestedness as $\mathrm{nest}_0(\mathbf{Ts}, \mathbf{T}) = \langle \mathbf{L}_{\xi_1}, \mu_{\xi_1}(\mathbf{L}_{\xi_1} \to \xi_1) \rangle = \langle \mathbf{L}_{\xi_1}, \mathbf{Nt} \rangle$, where $\mathbf{L}_{\xi_1} \coloneqq \mu_{\xi_0}(\xi_0, \xi_1 \to \xi_0 \to \xi_0)$. In the opposite direction, we can also introduce simultaneity as $\mathrm{simul}(\mathbf{L}_{\xi_1}, \mu_{\xi_1}(\mathbf{L}_{\xi_1} \to \xi_1)) = \mu_{\zeta,\xi_1}(\zeta, \xi_1 \to \zeta \to \zeta, \zeta \to \xi_1) = (\mathbf{Ts}, \mathbf{T})$.

2.1.3 Partial Continuous Functionals

Now, we think about concrete information systems. For a type ρ the information system IS_ρ is defined to be a triple $\langle \mathrm{Tok}_\rho, \mathrm{Con}_\rho, \vdash_\rho \rangle$. The partial continuous functionals of type ρ are defined by the ideals $|\mathrm{IS}_\rho|$. The partial continuous functionals of type $\rho \to \sigma$ correspond to the continuous functions from $|\mathrm{IS}_\rho|$ to $|\mathrm{IS}_\sigma|$. The treatment of information systems and ideals are due to Schwichtenberg [SW12].

Definition 2.1.23 (Information systems)**.** We simultaneously define Tok_ι, $\mathrm{Tok}_{\rho\to\sigma}$, Con_ι and $\mathrm{Con}_{\rho\to\sigma}$.

1. The tokens in Tok_ι are the type-correct constructor expressions $C_\iota\, \vec{a}^*$ where each a_i^* is an extended token, namely, a token or the special symbol $*$ which carries no information.

2. The tokens in $\mathrm{Tok}_{\rho\to\sigma}$ are the pairs of U in Con_ρ and a in Tok_σ.

3. The consistent set Con_ι is a finite set U of elements in Tok_ι such that for a specific constructor C, say of type $\tau_0 \to \cdots \to \tau_{n-1} \to \iota$, all elements of U are of the form $Ca_{i0} \cdots a_{in-1}$ satisfying that $\{a_{0m}, a_{1m} \cdots, a_{jm}\}$ are consistent for all $m < n$.

4. Let I be a finite set of indices. Pairs $\{(U_i, b_i) \,|\, i \in I\}$ are in $\mathrm{Con}_{\rho\to\sigma}$ if and only if $\forall_{J\subseteq I} \left(\bigcup_{j\in J} U_j \in \mathrm{Con}_\rho \to \{b_j \,|\, j \in J\} \in \mathrm{Con}_\sigma \right)$.

We define $\vdash_\iota$ and $\vdash_{\rho\to\sigma}$.

1. For any U, $U \vdash_\rho *$ and $U \vdash_{\rho\to\sigma} *$.

2. Let C be a constructor, then $\{C\vec{a}_0^*, \ldots, C\vec{a}_{n-1}^*\} \vdash_\iota C\vec{a}^*$ if and only if $\{a_{0m}, \ldots, a_{n-1\,m}\} \vdash a_m$.

Based on information systems developed so far, we define partial continuous functionals to be ideals.

Definition 2.1.24 (Partial continuous functionals)**.** Let IS_ρ be the information system $\langle \mathrm{Tok}_\rho, \mathrm{Con}_\rho, \vdash_\rho \rangle$ of type ρ. The set of partial continuous functionals of type ρ is defined to be the set of ideals in IS_ρ.

A partial continuous functional x is *computable* if x is a recursively enumerable set of tokens. Among ideals, we define cototal and total ideals in which we are especially interested. We give an explicit definition of cototal and total ideals of finitary algebras. In general, we define cototality and totality predicates in Section 2.3.

Definition 2.1.25 (One-step extension)**.** Consider a constructor expression with at least one occurrence of the special token $*$. We denote $P(*)$ to mean such a constructor expression one of whose occurrences of $*$ is pointed. An arbitrary $P(C\vec{a}^*)$ is *one-step extension* of $P(*)$, written $P(C\vec{a}^*) \succ_1 P(*)$. An occurrence of $*$ is *non-parametric* if the path from the root to this $*$ does not go through a parameter argument of a constructor.

Example 2.1.26 (One-step extension in **N** and **L**)**.** Assume algebras **N** and **L**(**N**). Let $P_1(*)$ be $\mathsf{S}*$ in **N**, then $P_1(\mathsf{S}*)$, namely, $\mathsf{S}(\mathsf{S}*)$ and also $P_1(\mathsf{0})$, namely, $\mathsf{S}\,\mathsf{0}$ are one-step extensions, written $P_1(*) \succ_1 P_1(\mathsf{S}*)$ and $P_1(*) \succ_1 P_1(\mathsf{0})$, respectively. Let $P_2(*)$ be $*::*$ in **L**(**N**), where the first occurrence of $*$ is pointed, then $P_2(*) \succ_1 P_2(\mathsf{0})$. Let P_3 be $*::*$, where the second occurrence of $*$ is pointed, then $P_3(*) \succ_1 P_3(*::*)$. The occurrences of $*$ in P_1 and P_3 are non-parametric.

Definition 2.1.27 (Cototal ideals and total ideals of finitary algebras)**.** Let ι be an algebra and IS_ι be the information system of ι. An ideal x of IS_ι is *cototal* if every constructor tree $P(*)$ in x has a $\succ_1$-predecessor $P(C\vec{*})$ in x. A cototal ideal x is total if the relation $\succ_1$ on x is well founded. An ideal is called *structure-cototal* (*structure-total*) if $\succ_1$ is defined with respect to $P(*)$ with a non-parametric distinguished occurrence of $*$.

Example 2.1.28 (Ideals of $\mathrm{IS}_{\mathbf{N}}$)**.** Consider tokens and entailment for **N** as in Figure 2.1. A line also means one-step extension of nodes, namely, viewed as tokens. An arbitrary path from a root, $\mathsf{0}$ or $\mathsf{S}*$, forms an ideal. If a path reaches a leaf, which is a finite constructor expression with no $*$, a set of tokens on this path is a total ideal. If a path goes from $\mathsf{S}*$ to the right branch, a set of tokens on this path is a cototal ideal $\{\mathsf{S}^n\ *\mid n \geq 1\}$.

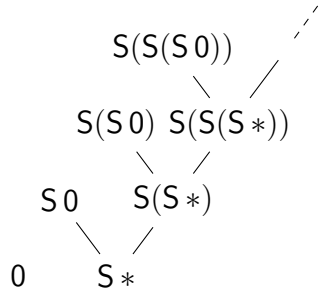

Figure 2.1: Tokens and entailment in **N**

Every constructor symbol C of algebra ι generates the ideal in the function space by $r_C \coloneqq \{(\vec{U}, C\vec{a}^*) \mid \vec{U} \vdash \vec{a}^*\}$. Let ι, $\vec{\iota}$ and $\vec{\iota}''$ be algebras. If $\iota, \vec{\iota} = \mathrm{nest}_n(\vec{\iota}'')$ for some n or $\vec{\iota}'' = \mathrm{simul}(\iota, \vec{\iota})$, the domains $|\mathrm{IS}_\iota|$ and $|\mathrm{IS}_{\iota'_0}|$, and the domains $|\mathrm{IS}_{\iota_i}|$ and $|\mathrm{IS}_{\iota'_{i+1}}|$ for each i are *isomorphic*. We formulate the domain isomorphism by means of approximable maps in the following way.

Definition 2.1.29 (Deductive closure)**.** We define the deductive closure of $U \in \mathrm{Con}$ to be $\overline{U} \coloneqq \{a \mid U \vdash a\}$.

Definition 2.1.30 (Domain isomorphism)**.** Let ι and ι' be algebras. The domains IS_ι and $\mathrm{IS}_{\iota'}$ are isomorphic if there exists an approximable map $r \subseteq \mathrm{Con}_\iota \times \mathrm{Tok}_{\iota'}$ such that the following holds:

1. $\forall_{U \in \mathrm{Con}_\iota}, \exists_{V \in \mathrm{Con}_{\iota'}}, U^r = \overline{V}$,

2. $\forall_{V\in\mathrm{Con}_{\iota'}}, \exists_{U\in\mathrm{Con}_{\iota}}, \overline{V} = U^r$,
3. $\forall_{U,V\in\mathrm{Con}_{\iota}}, U^r \subseteq V^r \to V \vdash U$,

where for $U \in \mathrm{Con}_\iota$ we abbreviate $\{b \in \mathrm{Tok}_{\iota'} \mid U\ r\ b\}$ as U^r.

Proposition 2.1.31 (Domain isomorphism between nestedness and simultaneity)**.** *Let $\vec{\iota}$ and $\vec{\iota}'$ be algebras of length l. If $nest_n(\vec{\iota}) = \langle \iota'_n, \vec{\iota}' - \iota'_n\rangle$ for some $n < l$, where $\vec{\iota}' - \iota'_n$ is obtained by removing ι'_n from $\vec{\iota}'$, then $|\mathrm{IS}_{\iota_i}|$ is isomorphic to $|\mathrm{IS}_{\iota'_i}|$ for each $i < l$. If $simul(\iota'_n, \vec{\iota}' - \iota'_n) = (\iota_n, \vec{\iota} - \iota_n)$, then $|\mathrm{IS}_{\iota_i}|$ is isomorphic to $|\mathrm{IS}_{\iota'_i}|$ for each $i < l$.*

Proof. For each $m < l$ we denote the constructors of algebras ι_m by $\vec{C}_m$ of length k_m and the constructors of algebras ι'_m by $\vec{C}'_m$ of length k_m. We define bijections ϕ_m from a token of ι_m to a token of ι'_m.

$$\phi_m(*) \coloneqq *, \qquad \phi_m(C_{mj}(\vec{a}^*)) \coloneqq C'_{mj}(\phi(a^*_\nu))_{\nu<n_m} \text{ for each } j < k_m.$$

For each $m < l$ we define $r_m \subseteq \mathrm{Con}_{\iota'_m} \times \mathrm{Tok}_{\iota_m}$ to be $U\ r_m\ b' \coloneqq \overline{U} \ni \phi_m^{-1}(b')$, where r_m is an approximable map. We prove the three conditions of domain isomorphisms. For the first one, we assume $U \in \mathrm{Con}_{\iota_m}$ and let V be $\{\phi_m(b) \mid b \in U\}$ which is in $\mathrm{Con}_{\iota'_m}$. For the second one, we use ϕ^{-1} instead. For the third one, we assume $U, V \in \mathrm{Con}_{\iota_m}$ and $U^r \subseteq V^r$, namely, $\forall_b(b \in \overline{U} \to b \in \overline{V})$. We prove $V \vdash U$ which is $\forall_{b\in U}(V \vdash b)$. Assume $b \in U$. By the assumption it implies $b \in \overline{V}$, hence $V \vdash b$. □

Example 2.1.32 (Isomorphism between $(\mathbf{Ts}, \mathbf{T})$ and $(\mathbf{L_{Nt}}, \mathbf{Nt})$)**.** Assume $(\mathbf{Ts}, \mathbf{T})$ and $\mathbf{Nt}$. The domains $|\mathrm{IS}_{\mathbf{Ts}}|$ and $|\mathrm{IS}_{\mathbf{L_{Nt}}}|$ are isomorphic and also $|\mathrm{IS}_{\mathbf{T}}|$ and $|\mathrm{IS}_{\mathbf{Nt}}|$ are.

2.2 T^+ as an Extension of Gödel's T

We study the syntactic aspect of computation through a formal term language. By extending Gödel's T [Göd58], we define a typed calculus T^+. In addition to *constructors* and *recursion operators* of the original T, we have *destructors*, *corecursion operators* and *programmable constants*. One can define *computation rules* and also *rewriting rules* for a programmable constant. The syntax of T^+ is the following.

Definition 2.2.1 (T^+)**.** Let ρ, σ be types and x a variable. Define terms as follows

$$M, N \coloneqq x^\rho \mid (\lambda_{x^\rho} M^\sigma)^{\rho\to\sigma} \mid (M^{\rho\to\sigma} N^\rho)^\sigma \mid \mathrm{C}^\rho$$

Here C^ρ is a constant, either a constructor, a recursion operator, a destructor, a corecursion operator or a programmable constant. We study them in the next sections. Following the standard way, we define the β-conversion.

Definition 2.2.2 (Free variables)**.** We define a set of free variables $\mathrm{FV}(M)$ for a term M by induction on the construction of the term.

$$\mathrm{FV}(x) \coloneqq \{x\}, \qquad \mathrm{FV}(MN) \coloneqq \mathrm{FV}(M) \cup \mathrm{FV}(N),$$
$$\mathrm{FV}(\mathrm{C}) \coloneqq \emptyset, \qquad \mathrm{FV}(\lambda_x N) \coloneqq \mathrm{FV}(N) \setminus \{x\}.$$

The term substitution $M[x/N]$ is intuitively the result of replacing every occurrence of x in M by N. Formally we define it in the following way with avoiding the variable capturing.

Definition 2.2.3 (Term substitution)**.** For terms M, N we define the term substitution $M[x/N]$ by induction on the construction of M. In the last case we suppose that a fresh variable z is available.

$$
\begin{aligned}
&x[x/N] \coloneqq N,\\
&\mathrm{C}[x/N] \coloneqq \mathrm{C},\\
&z[x/N] \coloneqq z \text{ where } z \text{ is a variable with } z \neq x,\\
&(M_0M_1)[x/N] \coloneqq M_0[x/N](M_1[x/N]),\\
&(\lambda_x M)[x/N] \coloneqq \lambda_x M,\\
&(\lambda_y M)[x/N] \coloneqq \lambda_y M[x/N] \text{ if } y \notin \mathrm{FV}(M),\\
&(\lambda_y M)[x/N] \coloneqq (\lambda_z M[y/z])[x/N] \text{ if } y \in \mathrm{FV}(M).
\end{aligned}
$$

Definition 2.2.4 (β-conversion)**.** $(\lambda_x M)N \mapsto M[x/N]$.

In the rest of this section, we formally develop T^+. Constructors and recursion operators are given in Section 2.2.1, corecursion operators and destructors are given in Section 2.2.2 and programmable constants are given in Section 2.2.3.

2.2.1 Constructors and Recursion Operators

For each constructor type of algebra, there is a constructor symbol. We adopt each constructor symbol as a special constant called *constructors*.

Definition 2.2.5 (Constructors)**.** Let ι be an algebra $\mu_\xi((\rho_{i\nu}(\xi))_{\nu<n_i} \to \xi)_{i<k}$. For each constructor type of ι a constructor C_i is defined as a constant of type $(\rho_{i\nu}(\iota))_{\nu<n_i} \to \iota$.

Constructors are means to construct (tree structured) objects from the root to the leaves. On the other hand, we have recursion operators to inspect an object which is built by constructors. A recursion operator reads an object from its leaves to the root. We define for each algebra ι the recursion operator $\mathcal{R}_\iota^\tau$ where τ is an arbitrary type parameter. We first define *map operators* which are used in the definition of the conversion rules of recursion operators. Then we define recursion operators. Later we also use map operators to define corecursion operators.

Definition 2.2.6 (Map operator)**.** Let $\vec{\alpha}$ be a list of type variables of length $l \geq 0$, $\iota(\vec{\alpha})$ be an algebra $\mu_\xi((\rho_{i\nu}(\vec{\alpha},\xi))_{\nu<n_i} \to \xi)_{i<k}$ and $C_i^{\vec{\alpha}}$ be the i-th constructor of $\iota(\vec{\alpha})$. Assume a list of function types $\sigma_i \to \tau_i$ for $0 \leq i < l$, written $\vec{\sigma} \to \vec{\tau}$, where l is the length of $\vec{\alpha}$. Assuming $\rho(\vec{\alpha})$ and $\vec{\pi}(\vec{\alpha})$ are a type and a list of types of length k, respectively, for where

the occurrences of $\vec{\alpha}$ in $\rho(\vec{\alpha})$ and $\pi_j(\vec{\alpha})$ for $j < k$ are strictly positive, we define the map operator $\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\rho(\vec{\alpha})}$ by induction on the construction of ρ.

$$\begin{aligned}
&\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\rho(\vec{\alpha})} : \rho(\vec{\sigma}) \to (\sigma_i \to \tau_i)_{0\le i<l} \to \rho(\vec{\tau}),\\
&\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}(\beta\to\rho(\vec{\alpha}))} N^{\beta\to\rho(\vec{\sigma})}\vec{M}K^{\beta} \mapsto \mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\rho(\vec{\alpha})}(NK)^{\rho(\vec{\sigma})}\vec{M},\\
&\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\alpha_i} N^{\sigma_i}\vec{M} \mapsto M_i N,\\
&\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\rho} N^{\rho}\vec{M} \mapsto N \qquad \text{if } \mathrm{FV}(\rho)\cap\vec{\alpha} = \emptyset,\\
&\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\iota_{\vec{\pi}(\vec{\alpha})}}(C_i^{\vec{\pi}(\vec{\sigma})}\vec{N})^{\iota_{\vec{\pi}(\vec{\sigma})}}\vec{M} \mapsto C_i^{\vec{\pi}(\vec{\tau})}(\mathcal{M}^{\vec{\pi}(\vec{\sigma}),\iota_{\vec{\pi}(\vec{\sigma})}\to\vec{\pi}(\vec{\tau}),\iota_{\vec{\pi}(\vec{\tau})}}_{\lambda_{\vec{\beta},\gamma}\rho_{i\nu}(\vec{\beta},\gamma)} N_\nu\vec{K}(\mathcal{M}^{\vec{\pi}(\vec{\sigma})\to\vec{\pi}(\vec{\tau})}_{\iota}\cdot\vec{K}))_{\nu<n_i},
\end{aligned}$$

where $\mathcal{M}\cdot\vec{M}$ stands for $\lambda_x(\mathcal{M}x\vec{M})$ and $K_j \coloneqq \mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\pi_j(\vec{\alpha})}\cdot\vec{M}$ for $j < |\vec{\pi}|$.

We define the type and the conversion rule of recursion operators.

Definition 2.2.7 (Recursion operator)**.** Let ι be an algebra $\mu_\xi((\rho_{i\nu}(\vec{\alpha},\xi))_{\nu<n_i}\to\xi)_{i<k}$. For an arbitrary type τ, the type of the recursion operator $\mathcal{R}^\tau_\iota$ is defined to be $\iota\to\vec{\delta}\to\tau$ where $\delta_i \coloneqq (\rho_{i\nu}(\vec{\alpha},\iota\times\tau))_{\nu<n_i}\to\tau$.

The conversion rule of the recursion operators is

$$\mathcal{R}^\tau_\iota(C_i\vec{N})\vec{M} \mapsto M_i(\mathcal{M}^{\iota\to\iota\times\tau}_{\lambda_\xi\rho_{i\nu}(\vec{\alpha},\xi)}\ N_\nu\ \lambda_z\langle z,\mathcal{R}^\tau_\iota z\vec{M}\rangle)_{\nu<n_i}.$$

We call each δ_i a *step type* and each M_i a *step term*. If $\rho_{i\nu}(\vec{\alpha},\xi)$ is just ξ, we conventionally type such a part of the step type δ_i by currying $\iota\times\tau$, namely, with duplicating ι and τ. Then the conversion rule is given without a map operator, again by duplicating the arguments N_i and $\mathcal{R}^\tau_\iota N_i\vec{M}$ in the step term M_i. Recursion operator $\mathcal{R}_\iota$ makes a case distinction on the outermost constructor of the first argument. Assuming the i-th constructor is the case, then the i-th step term is called with arguments including recursive calls of the recursion operator. An argument consists of two parts, the previous input and the previous output to the recursion operator, which are computed from the argument N_ν of the constructor by means of a map operator.

Example 2.2.8 (Type of $\mathcal{R}^\tau_{\mathbf{L}_\alpha}$)**.** Consider the type of $\mathcal{R}^\tau_{\mathbf{L}_\alpha}$. In the form of constructor types, $n_0 = 0$, $n_1 = 2$, $\rho_{10} = \alpha$ and $\rho_{11} = \xi$ specify the above constructor types. Since $n_0 = 0$, no $\rho_{0\nu}$ occurs, thus $\delta_0 = \tau$. Next, $\delta_1 = \alpha\to(\mathbf{L}_\alpha\times\tau)\to\tau$. As mentioned, here we use $\alpha\to\mathbf{L}_\alpha\to\tau\to\tau$ instead of the above step type with products by duplicating $\mathbf{L}_\alpha\times\tau$. Therefore, the type of $\mathcal{R}^\tau_{\mathbf{L}_\alpha}$ is $\mathbf{L}_\alpha\to\tau\to(\alpha\to\mathbf{L}_\alpha\to\tau\to\tau)\to\tau$.

Example 2.2.9 (Descending list of natural numbers)**.** The recursion operator on natural numbers, $\mathcal{R}^\tau_{\mathbf{N}}$, is given as

$$\begin{aligned}
&\mathcal{R}^\tau_{\mathbf{N}} : \mathbf{N}\to\tau\to(\mathbf{N}\to\tau\to\tau)\to\tau,\\
&\mathcal{R}^\tau_{\mathbf{N}}\mathsf{0}M_0M_1 \mapsto M_0,\\
&\mathcal{R}^\tau_{\mathbf{N}}(\mathsf{S}n)M_0M_1 \mapsto M_1 n(\mathcal{R}^\tau_{\mathbf{N}}nM_0M_1).
\end{aligned}$$

The following term $Desc$ computes a list of natural numbers descending from the given natural number to 1.

$$\begin{aligned} &Desc : \mathbf{N} \to \mathbf{L_N} \\ &Desc \coloneqq \lambda_n(\mathcal{R}_{\mathbf{N}}\ n\ []\ \lambda_{m,l}(\mathsf{S}m::l)). \end{aligned}$$

The computation of $Desc$ is as follows:

$$Desc\,\mathsf{0} \mapsto [], \qquad Desc(\mathsf{S}n) \mapsto \mathsf{S}n :: Desc\,n.$$

Example 2.2.10 (Recursion operator on $\mathbf{Nt}$)**.** The recursion operator on $\mathbf{Nt}$ is given as follows.

$$\begin{aligned} &\mathcal{R}^\tau_{\mathbf{Nt}} : \mathbf{Nt} \to (\mathbf{L}_{\mathbf{Nt}\times\tau} \to \tau) \to \tau \\ &\mathcal{R}^\tau_{\mathbf{Nt}}(\mathsf{Br}\,as)M \mapsto M(\mathcal{M}^{\mathbf{Nt}\to\mathbf{Nt}\times\tau}_{\lambda_\alpha \mathbf{L}_\alpha} as\lambda_a\langle a, \mathcal{R}aM\rangle) \end{aligned}$$

An important variant of recursion operators is *case operators*. It is to make a case distinction on the outermost constructor.

Definition 2.2.11 (Case operator)**.** Under the same assumptions in Definition 2.2.7, we define $\mathcal{C}^\tau_\iota$ to be a constant of type $\iota \to \vec{\sigma} \to \tau$, where $\sigma_i \coloneqq (\rho_{i\nu}(\vec{\alpha}, \iota))_{\nu<n_i} \to \tau$. The conversion rule of the case operators is

$$\mathcal{C}^\tau_\iota\,(C_i\vec{N})\,\vec{M} \mapsto M_i\,\vec{N}.$$

Following a common style found in generic programming languages, we adopt to denote a term $\mathcal{C}^\tau_\iota\,t^\iota\,\vec{M}$ also in the following way.

$$\begin{array}{ccc} \text{Case } t^\iota \text{ of } C_0\vec{x}_0 \to L_0 & & \text{Case } t^\iota \text{ of } \lambda_{\vec{x}_0} L_0 \\ \vdots & \text{or} & \vdots \\ C_{k-1}\vec{x}_{k-1} \to L_{k-1} & & \lambda_{\vec{x}_{k-1}} L_{k-1}, \end{array}$$

where $x_{i\nu}$ is a fresh variable of type $\rho_{i\nu}(\vec{\alpha}, \iota)$ for $\nu < n_i$ and L_i is a term such that $M_i\vec{x}_i \mapsto^* L_i$. We can omit $\vec{x}_i$ when $n_i = 0$. A line break can be replaced by a semicolon (;).

A well-known programming construction of if-then-else is given by $\mathcal{C}^\tau_{\mathbf{B}}$ of type $\mathbf{B} \to \tau \to \tau \to \tau$. We accept to denote $\mathcal{C}^\tau_{\mathbf{B}}\,M\,N_0\,N_1$ as follows.

$$\textbf{if } M \textbf{ then } N_0 \textbf{ else } N_1$$

We study simultaneous recursion operators.

Definition 2.2.12 (Simultaneous recursion operator)**.** Let $\vec{\tau}$ and $\vec{\xi}$ be lists of types of length $l > 0$ and $\vec{\iota}$ be simultaneous algebras $\mu_{\vec{\xi}}((\rho_{i\nu}(\vec{\xi}))_{\nu<n_i} \to \xi_{j_i})_{i<k}$. We define simultaneous

recursion operators on $\vec{\iota}$. For each $m < l$ let σ_m be $\iota_m \times \tau_m$ and the step types δ_{j_i} be $(\rho_{i\nu}(\vec{\sigma}))_{\nu<n_i} \to \tau_{j_i}$, then the types of recursion operators are defined for $m < l$ as follows.

$$\mathcal{R}^{\vec{\tau}}_{\vec{\iota},m} : \iota_m \to \vec{\delta} \to \tau_m.$$

The conversion rules are

$$\mathcal{R}^{\vec{\tau}}_{\vec{\iota},j_i}(C_i\vec{N})\vec{M} \mapsto M_i(\mathcal{M}^{\vec{\iota}\to\vec{\sigma}}_{\lambda_{\vec{\alpha}}\rho_{i\nu}(\vec{\alpha})}N_\nu(\lambda_x\langle x, \mathcal{R}^{\vec{\tau}}_{\vec{\iota},m}x\vec{M}\rangle)_{m<l})_{\nu<n_i}.$$

Example 2.2.13 (Recursion operator on $(\mathbf{Ts}, \mathbf{T})$)**.** Consider simultaneous recursion operators $\mathcal{R}^{\vec{\tau}}_{(\mathbf{Ts},\mathbf{T}),0}$ and $\mathcal{R}^{\vec{\tau}}_{(\mathbf{Ts},\mathbf{T}),1}$ which are non-nested. For $0 \le i < 3$, let δ_i be

$$\delta_0 \coloneqq \tau_0, \qquad \delta_1 \coloneqq \mathbf{T} \to \tau_1 \to \mathbf{Ts} \to \tau_0 \to \tau_0, \qquad \delta_2 \coloneqq \mathbf{Ts} \to \tau_0 \to \tau_1.$$

The types of $\mathcal{R}^{\vec{\tau}}_{(\mathbf{Ts},\mathbf{T}),0}$ and $\mathcal{R}^{\vec{\tau}}_{(\mathbf{Ts},\mathbf{T}),1}$ are

$$\mathcal{R}^{\vec{\tau}}_{(\mathbf{Ts},\mathbf{T}),0} : \mathbf{Ts} \to \delta_0 \to \delta_1 \to \delta_2 \to \tau_0, \quad \mathcal{R}^{\vec{\tau}}_{(\mathbf{Ts},\mathbf{T}),1} : \mathbf{T} \to \delta_0 \to \delta_1 \to \delta_2 \to \tau_1.$$

The conversion rules are as follows.

$$\begin{aligned}
&\mathcal{R}^{\vec{\tau}}_{(\mathbf{Ts},\mathbf{T}),0}\,\mathsf{Empty}\,\vec{M} \mapsto M_0\\
&\mathcal{R}^{\vec{\tau}}_{(\mathbf{Ts},\mathbf{T}),0}\,(\mathsf{Tcons}\,a\,as)\,\vec{M} \mapsto M_1\,a\,(\mathcal{R}^{\vec{\tau}}_{(\mathbf{Ts},\mathbf{T}),1}a\vec{M})\,as\,(\mathcal{R}^{\vec{\tau}}_{(\mathbf{Ts},\mathbf{T}),0}as\vec{M})\\
&\mathcal{R}^{\vec{\tau}}_{(\mathbf{Ts},\mathbf{T}),1}\,(\mathsf{Branch}\,as)\,\vec{M} \mapsto M_2\,as\,(\mathcal{R}^{\vec{\tau}}_{(\mathbf{Ts},\mathbf{T}),0}as\vec{M})
\end{aligned}$$

Example 2.2.14 (Substitution of trees)**.** We define a function of type $\mathbf{T} \to \mathbf{T} \to \mathbf{T}$ which replaces each leaves $\mathsf{Branch}\,\mathsf{Empty}$ in the first argument by the second one as follows.

$$\lambda_{a,b}(\mathcal{R}_{(\mathbf{Ts},\mathbf{T}),1}\,a\,b\,\lambda_{\_,p,\_,q}(\mathsf{Branch}(\mathsf{Tcons}\,p\,q))\,\lambda_{\_,p}p).$$

The underscore $(\_)$ is used as a place holder for an abstracted variable which is unused.

When only some algebras of simultaneously defined ones are needed to recur, simplified simultaneous recursion operators suffice. Generally, we assume $\mathcal{R}^{\vec{\tau}}_{\vec{\iota}}$ is defined and build a simplified one. We first specify what algebras $\iota_{i_0}, \ldots, \iota_{i_k}$ from $\vec{\iota}$ are *irrelevant*, that is, are not needed to recur. We drop all δ_i whose final value type is one of $\tau_{i_0}, \ldots, \tau_{i_k}$. From argument types of each remaining δ_i, we drop all $\iota_{i_0}, \ldots, \iota_{i_k}$ and $\tau_{i_0}, \ldots, \tau_{i_k}$. The remaining algebras are on the other hand called the *relevant algebras*.

For example, we simplify $\mathcal{R}^{\vec{\tau}}_{\mathbf{Ts},\mathbf{T}}$ in two ways.

1. No recursion on $\mathbf{T}$. Among $\vec{\delta}$, δ_2 and δ_3 are dropped because their final value type is τ_1 which is irrelevant. Remaining step types are δ_0 and δ_1. The types $\mathbf{T}$ and τ_1 occur in δ_0 no more. From δ_1 we remove the type arguments $\mathbf{T}$ and τ_1. The type and the conversion rules are

$$\begin{aligned}
&\mathcal{R}^{\tau}_{\mathbf{Ts}} : \mathbf{Ts} \to \tau \to (\mathbf{Ts} \to \tau \to \tau) \to \tau,\\
&\mathcal{R}^{\tau}_{\mathbf{Ts}}\mathsf{Empty}M_0M_1 \mapsto M_0, \quad \mathcal{R}^{\tau}_{\mathbf{Ts}}(\mathsf{Tcons}\,a\,as)M_0M_1 \mapsto M_1as(\mathcal{R}^{\tau}_{\mathbf{Ts}}asM_0M_1).
\end{aligned}$$

2. No recursion on **Ts**. As we did in 1., we remove irrelevant types. Then the type and the conversion rules are

$$\mathcal{R}^{\tau}_{\mathbf{T}} : \mathbf{T} \to \tau \to \tau, \qquad \mathcal{R}^{\tau}_{\mathbf{T}}(\mathsf{Branch}\, as) M_0 \mapsto M_0.$$

Example 2.2.15 (Length of a tree)**.** We define a function Lh to count the length of a tree, namely, the number of branches at the root node.

$$Lh : \mathbf{T} \to \mathbf{N}, \qquad Lh \coloneqq \lambda_a(\mathcal{C}\, a\, \lambda_{as}(\mathcal{R}^{\mathbf{N}}_{\mathbf{Ts}}\, as\, 0\, \lambda_{\_,n}(\mathsf{S}\, n))).$$

Due to the inhabitedness of types, for an arbitrary type ρ there is an *inhabitant* of type ρ, namely, a term of type ρ. We can find from the derivation of $\mathrm{Inhab}_{\vec{\zeta}}(\rho)$ an inhabitant of type ρ.

Definition 2.2.16 (Inhabitant)**.** Assume τ is a type, a derivation of $\mathrm{Inhab}_{\vec{\zeta}}(\tau)$ is given and an inhabitant c^{α} of each type variable $\alpha \in \mathrm{FV}(\tau)$ is given. We inductively define from the derivation of $\mathrm{Inhab}_{\vec{\zeta}}(\tau)$ an inhabitant of type τ. If $\mathrm{Inhab}_{\vec{\zeta}}(\alpha)$ is derived by the first rule, an inhabitant is c^{α}. If $\mathrm{Inhab}_{\vec{\zeta}}(\rho \to \sigma)$ is derived by the second rule, there is an inhabitant $\lambda_{x^{\rho}} c^{\sigma}$ of type $\rho \to \sigma$ where c^{σ} comes by induction hypothesis. If $\mathrm{Inhab}_{\vec{\zeta}}(\iota)$, where $\iota = \mu_{\xi}((\rho_{i\nu})_{\nu<n_i} \to \xi)_{i<k}$, is derived by the third rule, there is an inhabitant $C_i(c^{\rho_{i\nu}})_{\nu<n_i}$ where C_i is the i-th constructor of ι and each $c^{\rho_{i\nu}}$ for all $\nu < n_i$ comes by induction hypothesis. An inhabitant c^{ρ} is *canonical* if it is obtained from the canonical derivation of $\mathrm{Inhab}_{\vec{\zeta}}(\rho)$.

2.2.2 Destructors and Corecursion Operators

We study *destructors* and *corecursion operators* which are defined for each algebra. In contrast to constructors and recursion operators, here we deal with possibly non-well founded objects. Destructors are means to inspect an object. The inspection by destructors does not involve any repetition while the one of recursion operators does. On the other hand, corecursion operators provide means to construct an object. While one can build finite objects by constructors, one can build non-well founded objects by corecursion operators.

We take a look at an idea of the two new constants via an example in lists. Consider a term M of type $\mathbf{L}_{\alpha}$ constructed by the corecursion operator of $\mathbf{L}_{\alpha}$. When the destructor is applied to M, the result tells us either

1. that M behaves as an empty list and there is nothing to inspect, or

2. that M behaves as a cons of the head x and the tail M' which can be further inspected.

Using the destructor repeatedly, we can inspect M, M', M'', $\cdots$, then we observe x, x', x'', $\cdots$. It can be the case that the first case never happens. Through the observation by means of a destructor, a corecursion operator provides a reasonable way to form non-well founded objects. In fact our formulation of destructors and corecursion operators are slight different from what one expect from the above sketch: Our observation is done through

normalization without an explicit application of a destructor. An advantage is that our theory of abstract computability works also for the two constants due to cototal ideals.

We define finite products and finite sums of a list of types by generalizing the product algebra $\times$ and the sum algebra $+$. Intuitively the finite product is as an algebra $\mu_\xi(\alpha_0 \to \alpha_1 \to \ldots \to \alpha_{n-1} \to \xi)$, and the finite sum is as $\mu_\xi(\alpha_0 \to \xi, \alpha_1 \to \xi, \ldots, \alpha_{n-1} \to \xi)$. Using them, the types of destructors and corecursion operators are given.

Definition 2.2.17 (Product type $\prod$)**.** Let $\vec{\tau}$ be a list of types. We define $\prod \vec{\tau}$ by induction.

$$\prod() \coloneqq \mathbf{U}, \qquad \prod(\tau) \coloneqq \tau, \qquad \prod(\tau, \vec{\tau}) \coloneqq \tau \times \prod \vec{\tau}.$$

Let m be the length of $\vec{\tau}$. We define $\prod_{i<m} \tau_i$ to be $\prod \vec{\tau}$.

Definition 2.2.18 (Sum type $\sum$)**.** We first define $\alpha_0 \oplus \alpha_1$ as follows

$$\begin{aligned} &\mathbf{U} \oplus \mathbf{U} \coloneqq \mathbf{B}, && \beta_0 \oplus \mathbf{U} \coloneqq \mathbf{Ysumu}_{\beta_0}, \\ &\mathbf{U} \oplus \beta_1 \coloneqq \mathbf{Uysum}_{\beta_1}, && \beta_0 \oplus \beta_1 \coloneqq \beta_0 + \beta_1, \end{aligned}$$

where β_0 and β_1 are not $\mathbf{U}$ and $\mathbf{Ysumu}_\alpha$ is defined to be $\mu_\xi(\mathsf{JustL}^{\alpha \to \xi}, \mathsf{NoneR}^\xi)$. Let $\vec{\tau}$ be a list of types. We define $\sum \vec{\tau}$ by induction.

$$\sum() \coloneqq \mathbf{U}, \qquad \sum(\tau) \coloneqq \tau, \qquad \sum(\tau, \vec{\tau}) \coloneqq \tau \oplus \sum \vec{\tau}.$$

Let m be the length of $\vec{\tau}$. We define $\sum_{i<m} \tau_i$ to be $\sum \vec{\tau}$.

For readability we can abuse the sum algebra $\alpha + \beta$ instead of $\alpha \oplus \beta$ in the definition of the sum type. Then, we can write $\mathbf{U} + \alpha$ instead of $\mathbf{Uysum}_\alpha$, $\alpha + \mathbf{U}$ instead of $\mathbf{Ysumu}_\alpha$ and $\mathbf{U} + \mathbf{U}$ instead of $\mathbf{B}$. Corresponding to $\sum$ and $\prod$, we define the generalized pairing and injection. Then, we consider conversion rules of corecursion operator.

Definition 2.2.19 (Pairing)**.** Suppose that $\vec{t}$ is a non-empty list of terms, $\vec{\tau}$ is a non-empty list of types, where each t_i is of type τ_i and a natural number n is greater than 0 and smaller than or equal to the length of $\vec{t}$. We inductively define the pairing $\mathbf{pair}^{\vec{\tau}}\,\vec{t}$ of type $\prod_{i<n} \tau_i$.

$$\mathbf{pair}^{()}() \coloneqq \mathsf{U}, \qquad \mathbf{pair}^{(\tau)}(t) \coloneqq t, \qquad \mathbf{pair}^{(\tau,\vec{\tau})}(t, \vec{t}\,) \coloneqq \langle t, \mathbf{pair}^{\vec{\tau}}(\vec{t}\,)\rangle.$$

The superscript $\vec{\tau}$ can be omitted unless it makes a confusion.

We define the injection without using $\mathsf{InL}_{\alpha,\beta}$ and $\mathsf{InR}_{\alpha,\beta}$ provided $\alpha = \mathbf{U}$ or $\beta = \mathbf{U}$. If $\alpha = \beta = \mathbf{U}$, we use T and F instead of $\mathsf{InL}\,\mathsf{U}$ and $\mathsf{InR}\,\mathsf{U}$, respectively. Similarly we use None_β and $\mathsf{Just}_\beta\, t^\beta$ instead of $\mathsf{InL}_{\mathbf{U},\beta}\,\mathsf{U}$ and $\mathsf{InR}_{\mathbf{U},\beta}\, t$, and $\mathsf{JustL}_\alpha\, t^\alpha$ and NoneR_α instead of $\mathsf{InL}_{\alpha,\mathbf{U}}\, t$ and $\mathsf{InR}_{\alpha,\mathbf{U}}\,\mathsf{U}$.

Definition 2.2.20 (Injection)**.** Let $\vec{\tau}$ be a list of types and $m \leq 0$ be its length. We define the injection $\mathbf{in}^{\vec{\tau}}_{i<m}$. If $m = 0$, $\mathbf{in}^{()}_{i<0}$ is U of type $\mathbf{U}$. We can also write $\mathbf{in}^0$ instead. If $m > 0$,

let a term t be of type τ_i for $i < m$ and we define the injection $\mathbf{in}^{\vec{\tau}}_{i<m}$ of type $\tau_i \to \sum \vec{\tau}$. We first define $\mathbf{in}^{\langle \alpha_0,\alpha_1\rangle}_j$ of type $\alpha_j \to \alpha_0 \oplus \alpha_1$ for $j \in \{0,1\}$.

$$\begin{aligned}
&\mathbf{in}^{\langle \mathbf{U},\mathbf{U}\rangle}_0(t) \coloneqq \mathsf{T}, && \mathbf{in}^{\langle \mathbf{U},\beta_1\rangle}_0(t) \coloneqq \mathsf{None}_{\beta_1},\\
&\mathbf{in}^{\langle \beta_0,\mathbf{U}\rangle}_0(t) \coloneqq \mathsf{Just}_{\beta_1} t, && \mathbf{in}^{\langle \beta_0,\beta_1\rangle}_0(t) \coloneqq \mathsf{InL}_{\beta_0,\beta_1} t,\\
&\mathbf{in}^{\langle \mathbf{U},\mathbf{U}\rangle}_1(t) \coloneqq \mathsf{F}, && \mathbf{in}^{\langle \mathbf{U},\beta_1\rangle}_1(t) \coloneqq \mathsf{JustL}_{\beta_1} t,\\
&\mathbf{in}^{\langle \beta_0,\mathbf{U}\rangle}_1(t) \coloneqq \mathsf{NoneR}_{\beta_1}, && \mathbf{in}^{\langle \beta_0,\beta_1\rangle}_1(t) \coloneqq \mathsf{InR}_{\beta_0,\beta_1} t.
\end{aligned}$$

Then, the injection is as follows

$$\begin{aligned}
&\mathbf{in}^{(\tau)}_{i<1}(t) \coloneqq t, && \mathbf{in}^{(\tau,\vec{\tau})}_{0<n+1}(t) \coloneqq \mathbf{in}^{\langle \tau,\sum\vec{\tau}\rangle}_0(t),\\
&\mathbf{in}^{(\tau,\tau')}_{n<n+1}(t) \coloneqq \mathbf{in}^{\langle \tau,\tau'\rangle}_1(t), && \mathbf{in}^{(\tau,\vec{\tau})}_{i+1<n+1}(t) \coloneqq \mathbf{in}^{\langle \tau,\sum\vec{\tau}\rangle}_0(\mathbf{in}^{(\vec{\tau})}_{i<n}(t)) \text{ if } i < n.
\end{aligned}$$

Let m be the length of $\vec{\tau}$ We can write $\mathbf{in}^{\vec{\tau}}_i(t)$ instead of $\mathbf{in}^{\vec{\tau}}_{i<m}(t)$, and also $\mathbf{in}^m_i(t)$ if $\vec{\tau}$ can be omitted.

For readability, we can abuse constructors of $\mathbf{U}$ and $+$ in the following way to denote an injection. Let $\mathbf{in}^{\langle \alpha_0,\alpha_1\rangle}_j$ of type $\alpha_j \to \alpha_0 + \alpha_1$ for $j \in \{0,1\}$ be:

$$\begin{aligned}
&\mathbf{in}^{\langle \mathbf{U},\mathbf{U}\rangle}_0(t) \coloneqq \mathsf{InL}\,\mathsf{U}, && \mathbf{in}^{\langle \mathbf{U},\beta_1\rangle}_0(t) \coloneqq \mathsf{InL}\,\mathsf{U},\\
&\mathbf{in}^{\langle \beta_0,\mathbf{U}\rangle}_0(t) \coloneqq \mathsf{InL}\,t, && \mathbf{in}^{\langle \beta_0,\beta_1\rangle}_0(t) \coloneqq \mathsf{InL}\,t,\\
&\mathbf{in}^{\langle \mathbf{U},\mathbf{U}\rangle}_1(t) \coloneqq \mathsf{InR}\,\mathsf{U}, && \mathbf{in}^{\langle \mathbf{U},\beta_1\rangle}_1(t) \coloneqq \mathsf{InR}\,t,\\
&\mathbf{in}^{\langle \beta_0,\mathbf{U}\rangle}_1(t) \coloneqq \mathsf{InR}\,\mathsf{U}, && \mathbf{in}^{\langle \beta_0,\beta_1\rangle}_1(t) \coloneqq \mathsf{InR}\,t.
\end{aligned}$$

Corresponding to the generalized pairing and injection, we give the generalized projection and case distinction. Formally we can define them by means of the recursion operators on $\times$ and $+$ or programmable constants given in Section 2.2.3.

Definition 2.2.21 (Projection)**.** Suppose that $\vec{t}$ is a list of terms of length n whose types are $\vec{\tau}$. We define the projection π^n_k of type $\prod_{i<n}\tau_i \to \tau_k$ with the following conversion rule.

$$\pi^n_k(\mathbf{pair}^{\vec{\tau}}\,\vec{t}\,) \mapsto t_k.$$

We define case operators on a generalized sum type.

Definition 2.2.22 (Case operator on a generalized sum)**.** Suppose that $\vec{\alpha}$ is a list of types of length n. Let x be a variable of type $\sum_{i<n}\alpha_i =: \rho$ and M_i be a term of type $\alpha_i \to \tau$. We define a case operator $\mathcal{C}^\tau_\rho$ to be a constant of type $\rho \to (\alpha_i \to \tau)_{i<n} \to \tau$ with the conversion rule $\mathcal{C}(\mathbf{in}^n_i\,t)\vec{M} \mapsto M_i\,t$ for $i < n$. We also adopt a programming language style expression of $\mathcal{C}(\mathbf{in}^n_i\,t)\vec{M}$ as follows.

$$\begin{array}{lcl}
\text{Case } x \text{ of } \mathbf{in}^n_0\,u_0 \to L_0 & & \text{Case } x \text{ of } \lambda_{u_0} L_0\\
\quad\vdots & \text{or} & \quad\vdots\\
\mathbf{in}^n_{n-1}\,u_{n-1} \to L_{n-1} & & \lambda_{u_{n-1}} L_{n-1},
\end{array}$$

where u_i is a fresh variable of type α_i and L_i is a term such that $M_i\, u_i \mapsto^* L_i$. Instead of a line break, we can use ";" for a delimitor as in Example 2.2.29. It is converted into $M_i y$, provided x is of the form $\mathbf{in}_i^n\, y$ for some $i < n$. We also define a term $[M_0, \ldots, M_{n-1}]$ of type $\rho \to \tau$ to be $\lambda_x(\mathcal{C}\, x\, \vec{M})$.

We define destructors and corecursion operators.

Definition 2.2.23 (Types of destructor and corecursion operator)**.** Let ι be an algebra $\mu_\xi((\rho_{i\nu})_{\nu<n_i} \to \xi)_{i<k}$. We define constants the *destructor* on ι and the *corecursion operator* on ι with a type parameter τ, denoted by $\mathcal{D}_\iota$ and ${}^{\mathrm{co}}\mathcal{R}_\iota^\tau$, respectively. The type of $\mathcal{D}_\iota$ is

$$\iota \to \textstyle\sum_{i<k} \prod_{\nu<n_i} \rho_{i\nu}(\iota),$$

and the type of ${}^{\mathrm{co}}\mathcal{R}_\iota^\tau$ is

$$\tau \to \big(\tau \to \textstyle\sum_{i<k} \prod_{\nu<n_i} \rho_{i\nu}\big(\iota \oplus \tau\big)\big) \to \iota.$$

The type τ is called the *covalue type*.

Example 2.2.24 (Types of $\mathcal{D}_{\mathbf{L}_\alpha}$ and ${}^{\mathrm{co}}\mathcal{R}_{\mathbf{L}_\alpha}^\tau$)**.** Consider the type of $\mathcal{D}_{\mathbf{L}_\alpha}$. By Definition 2.2.18, $\mathbf{L}_\alpha \to \sum_{i<2} \prod_{\nu<n_i} \rho_{i\nu}(\alpha, \mathbf{L}_\alpha) = \mathbf{L}_\alpha \to \mathbf{Uysum}(\alpha \times \mathbf{L}_\alpha)$. Then, the type of $\mathcal{D}_{\mathbf{L}_\alpha}$ is informally denoted as $\mathbf{L}_\alpha \to \mathbf{U} + \alpha \times \mathbf{L}_\alpha$. Again by Definition 2.2.18, $\sum_{i<2} \prod_{\nu<n_i} \rho_{i\nu}(\alpha, \mathbf{L}_\alpha \oplus \tau) = \mathbf{Uysum}(\alpha \times (\mathbf{L}_\alpha \oplus \tau))$ provided $\tau \neq \mathbf{U}$. The type of ${}^{\mathrm{co}}\mathcal{R}_{\mathbf{L}_\alpha}^\tau$ is informally denoted as $\tau \to (\tau \to \mathbf{U} + \alpha \times (\mathbf{L}_\alpha + \tau)) \to \mathbf{L}_\alpha$. If $\tau = \mathbf{U}$, $\sum_{i<2} \prod_{\nu<n_i} \rho_{i\nu}(\alpha, \mathbf{Ysumu}(\mathbf{L}_\alpha)) = \mathbf{Uysum}(\alpha \times (\mathbf{Ysumu}(\mathbf{L}_\alpha)))$. The type of ${}^{\mathrm{co}}\mathcal{R}_{\mathbf{L}_\alpha}^{\mathbf{U}}$ is simplified and denoted as $\mathbf{U} + \alpha \times (\mathbf{L}_\alpha + \mathbf{U}) \to \mathbf{L}_\alpha$.

The conversion rule of the destructor is given as follows.

Definition 2.2.25 (Conversion rule of destructor)**.** Suppose that ι is an algebra defined to be $\mu_\xi((\rho_{i\nu}(\xi))_{\nu<n_i} \to \xi)_{i<k}$. The conversion rule of the destructor on ι is defined as follows.

$$\mathcal{D}_\iota(C_i \vec{t}\,) \mapsto \mathbf{in}_i^k(\mathbf{pair}(\vec{t}\,)).$$

The conversion rules of the corecursion operator are given as follows.

Definition 2.2.26 (Conversion rule of corecursion operator)**.** Let ι be $\mu_\xi((\rho_{i\nu}(\xi))_{\nu<n_i} \to \xi)_{i<k}$. For the ι and an arbitrary type τ, we define the conversion rule of ${}^{\mathrm{co}}\mathcal{R}_\iota^\tau$ as follows.

$$\begin{aligned} {}^{\mathrm{co}}\mathcal{R}_\iota^\tau \mapsto \lambda_{u,v}(\mathrm{Case}\ vu\ \mathrm{of}\ \mathbf{in}_0^k\, t &\to C_0(\mathcal{M}^{\iota+\tau\to\iota}_{\lambda_\beta \rho_{0\nu}(\beta)}(\pi_\nu^{n_0} t)[\mathrm{id}, {}^{\mathrm{co}}\mathcal{R}_\iota^\tau \cdot v])_{\nu<n_0} \\ &\vdots \\ \mathbf{in}_{k-1}^k\, t &\to C_{k-1}(\mathcal{M}^{\iota+\tau\to\iota}_{\lambda_\beta \rho_{(k-1)\nu}(\beta)}(\pi_\nu^{n_{k-1}} t)[\mathrm{id}, {}^{\mathrm{co}}\mathcal{R}_\iota^\tau \cdot v])_{\nu<n_{k-1}}). \end{aligned}$$

If $n_i = 0$, we take $\mathbf{in}_i^k\, \mathsf{U} \to C_i$ instead of a line in the above. We can omit the case distinction and the constructor immediately comes when $k = 1$.

We take a look at definitions of destructors on some algebras from Example 2.1.13, 2.1.14, 2.1.15 and 2.1.16.

Example 2.2.27 (Destructors on a simple algebra)**.**

$$\mathcal{D}_{\mathbf{U}} : \mathbf{U} \to \mathbf{U} \qquad \mathcal{D}_{\mathbf{B}} : \mathbf{B} \to \mathbf{U} + \mathbf{U}$$
$$\mathcal{D}_{\mathbf{U}}\mathsf{U} \mapsto \mathsf{U} \qquad \mathcal{D}_{\mathbf{B}}\mathsf{F} \mapsto \mathsf{InL}\,\mathsf{U}, \qquad \mathcal{D}_{\mathbf{B}}\mathsf{T} \mapsto \mathsf{InR}\,\mathsf{U}$$

$$\mathcal{D}_{\mathbf{N}} : \mathbf{N} \to \mathbf{U} + \mathbf{N} \qquad \mathcal{D}_{\mathbf{D}} : \mathbf{D} \to \mathbf{U} + (\mathbf{D} \times \mathbf{D})$$
$$\mathcal{D}_{\mathbf{N}}\mathsf{0} \mapsto \mathsf{InL}\,\mathsf{U} \qquad \mathcal{D}_{\mathbf{D}}\mathsf{0} \mapsto \mathsf{InL}\,\mathsf{U}$$
$$\mathcal{D}_{\mathbf{N}}(\mathsf{S}n) \mapsto \mathsf{InR}\,n \qquad \mathcal{D}_{\mathbf{D}}(\mathsf{D}d_0d_1) \mapsto \mathsf{InR}\langle d_0, d_1\rangle$$

Example 2.2.28 (Destructors on parameterized, nested and non-finitary algebras)**.**

$$\mathcal{D}_{\mathbf{L}_\alpha} : \mathbf{L}_\alpha \to \mathbf{U} + \alpha \times \mathbf{L}_\alpha \qquad \mathcal{D}_{\mathbf{Nt}} : \mathbf{Nt} \to \mathbf{L}_{\mathbf{Nt}}$$
$$\mathcal{D}_{\mathbf{L}_\alpha}[] \mapsto \mathsf{InL}\,\mathsf{U} \qquad \mathcal{D}_{\mathbf{Nt}}(\mathsf{Br}as) \mapsto as$$
$$\mathcal{D}_{\mathbf{L}_\alpha}(x{::}xs) \mapsto \mathsf{InR}\langle x, xs\rangle$$

$$\mathcal{D}_{\mathbf{O}} : \mathbf{O} \to \mathbf{U} + \mathbf{O} + (\mathbf{N} \to \mathbf{O})$$
$$\mathcal{D}_{\mathbf{O}}\mathsf{0} \mapsto \mathsf{InL}\,\mathsf{U}$$
$$\mathcal{D}_{\mathbf{O}}(\mathsf{S}x) \mapsto \mathsf{InR}(\mathsf{InL}x)$$
$$\mathcal{D}_{\mathbf{O}}(\mathsf{Sup}f) \mapsto \mathsf{InR}(\mathsf{InR}f)$$

We take a look at definitions of corecursion operators on some algebras from Example 2.1.13, 2.1.14, 2.1.15 and 2.1.16.

Example 2.2.29 (Corecursion operators on a simple algebra)**.**

$${}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbf{U}} : \tau \to (\tau \to \mathbf{U}) \to \mathbf{U} \qquad {}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbf{B}} : \tau \to (\tau \to \mathbf{B} + \mathbf{B}) \to \mathbf{B}$$
$${}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbf{U}} \mapsto \lambda_{u,v}(\mathrm{Case}\ vu\ \mathrm{of}\ \mathsf{U} \to \mathsf{U}) \qquad {}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbf{B}} \mapsto \lambda_{u,v}(\mathrm{Case}\ vu\ \mathrm{of}\ \mathsf{InL}\,\mathsf{U} \to \mathsf{InL}\,\mathsf{U}$$
$$\mathsf{InR}\,\mathsf{U} \to \mathsf{InR}\,\mathsf{U})$$
$${}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbf{N}} : \tau \to (\tau \to \mathbf{U} + (\mathbf{N} + \tau)) \to \mathbf{N}$$
$${}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbf{N}} \mapsto \lambda_{u,v}(\mathrm{Case}\ vu\ \mathrm{of}\ \mathsf{InL}\,\mathsf{U} \to \mathsf{0}; \mathsf{InR}\,x \to \mathsf{S}(\mathcal{M}x[\mathrm{id}, \lambda_z({}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbf{N}}zv)]))$$
$${}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbf{D}} : \tau \to (\tau \to \mathbf{U} + (\mathbf{D} + \tau) \times (\mathbf{D} + \tau)) \to \mathbf{D}$$
$${}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbf{D}} \mapsto \lambda_{u,v}(\mathrm{Case}\ vu\ \mathrm{of}\ \mathsf{InL}\,\mathsf{U} \to \mathsf{0}; \mathsf{InR}\,x \to \mathsf{D}(\mathcal{M}(\pi_0 x)[\mathrm{id}, \lambda_z({}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbf{D}}zv)])$$
$$(\mathcal{M}(\pi_1 x)[\mathrm{id}, \lambda_z({}^{\mathrm{co}}\mathcal{R}^\tau_{\mathbf{D}}zv)]))$$

Example 2.2.30 (Corecursion operators on parameterized, nested and non-finitary alge-

bras).

$$
\begin{aligned}
&{}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{L}_\alpha} : \tau \to (\tau \to \mathbf{U} + \alpha \times (\mathbf{L}_\alpha + \tau)) \to \mathbf{L}_\alpha\\
&{}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{L}_\alpha} \mapsto \lambda_{u,v}(\text{Case } vu \text{ of } \mathsf{InL}\,\mathsf{U} \to []; \mathsf{InR}\, y \to \pi_0 y :: \mathcal{M}(\pi_1 y)[\mathrm{id}, \lambda_z({}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{L}_\alpha} zv)])\\
&{}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{Nt}} : \tau \to (\tau \to \mathbf{L}_{\mathbf{Nt}+\tau}) \to \tau\\
&{}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{Nt}} \mapsto \lambda_{u,v}(\mathsf{Br}(\mathcal{M}^{\mathbf{Nt}+\tau\to\mathbf{Nt}}_{\lambda_\alpha \mathbf{L}_\alpha}(vu)[\mathrm{id}, \lambda_z({}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{Nt}} zv)]))\\
&{}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{O}} : \tau \to (\tau \to \mathbf{U} + (\mathbf{O} + \tau) + (\mathbf{N} \to \mathbf{O} + \tau)) \to \mathbf{O}\\
&{}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{O}} \mapsto \lambda_{u,v}(\text{Case } vu \text{ of } \mathsf{InL}\,\mathsf{U} \to \mathsf{0}\\
&\qquad\qquad\qquad\qquad \mathsf{InR}(\mathsf{InL}\, y) \to \mathsf{S}(\mathcal{M} y[\mathrm{id}, \lambda_z({}^{\mathrm{co}}\mathcal{R}^{\tau}_{\mathbf{O}} zv)])\\
&\qquad\qquad\qquad\qquad \mathsf{InR}(\mathsf{InR}\, f) \to \mathsf{Sup}(\mathcal{M} f[\mathrm{id}, \lambda_z({}^{\mathrm{co}}\mathcal{R}_{\mathbf{O}} zv)]))
\end{aligned}
$$

Destructors and corecursion operators are also defined on simultaneous algebras. In the definition of destructors, we collect the relevant types of constructors whose value type is same as the argument type of the destructor.

Definition 2.2.31 (Destructors and corecursion operators on simultaneous algebras)**.** Let $\vec{\iota}$ be simultaneous algebras $\mu_{\vec{\xi}}((\rho_{i\nu}(\vec{\xi}))_{\nu<n_i} \to \xi_{j_i})_{i<k}$ of length l. For each $m < l$, we can find a list of indices $\vec{r}_m$, such that $j_i = m$ if and only if i is in $\vec{r}_m$. For each $m < l$ let i be in $\vec{r}_m$ and $\hat{\imath}$ be an index to get i from $\vec{r}_m$, namely, $i = r_{m\hat{\imath}}$. The type and the conversion rule of the destructor are defined to be

$$\mathcal{D}_{\vec{\iota},m} : \iota_m \to \textstyle\sum_{i<|\vec{r}_m|} \prod_{\nu<n_i} \rho_{r_{mi}\nu}(\vec{\iota}), \qquad \mathcal{D}_{\vec{\iota},m}(C_i \vec{t}) \mapsto \mathbf{in}_{\hat{\imath}}^{|\vec{r}_m|}(\mathbf{pair}\, \vec{t}).$$

Let σ_i be a sum type $\iota_i + \tau_i$. For each $m < l$ we define the m-th costep type δ_m to be $\tau_i \to \sum_{i<|\vec{r}_m|} \prod_{\nu<n_i} \rho_{i\nu}(\vec{\sigma})$. The type of the m-th corecursion operator ${}^{\mathrm{co}}\mathcal{R}^{\vec{\tau}}_{\vec{\iota},m}$ is $\tau_m \to \vec{\delta} \to \iota_m$. Its conversion rule is given as

$$
\begin{aligned}
{}^{\mathrm{co}}\mathcal{R}^{\vec{\tau}}_{\vec{\iota},m} \mapsto \lambda_{u,\vec{v}}(&\text{Case } v_m u \text{ of}\\
&\mathbf{in}_0^{k'} t \to C_0(\mathcal{M}^{\vec{\sigma}\to\vec{\iota}}_{\lambda_{\vec{\beta}} \rho_{0\nu}(\vec{\beta})}(\pi_\nu^{n_0} t)[\mathrm{id}, {}^{\mathrm{co}}\mathcal{R}^{\vec{\tau}}_{\vec{\iota},m} \cdot \vec{v}]_{m<l})_{\nu<n_0}\\
&\quad\vdots\\
&\mathbf{in}_{k'-1}^{k'} t \to C_{k'-1}(\mathcal{M}^{\vec{\sigma}\to\vec{\iota}}_{\lambda_{\vec{\beta}} \rho_{(k'-1)\nu}(\vec{\beta})}(\pi_\nu^{n_{k'-1}} t)[\mathrm{id}, {}^{\mathrm{co}}\mathcal{R}^{\vec{\tau}}_{\vec{\iota},m} \cdot \vec{v}]_{m<l})_{\nu<n_{k'-1}}),
\end{aligned}
$$

where $k' \coloneqq |\vec{r}_m|$.

We take a look at an example of destructors and corecursion operators on simultaneous algebras.

Example 2.2.32 (Destructors on simultaneous algebras)**.**

$$
\begin{aligned}
&\mathcal{D}_{(\mathbf{Ts},\mathbf{T}),0} : \mathbf{Ts} \to \mathbf{U} + \mathbf{T} \times \mathbf{Ts} && \mathcal{D}_{(\mathbf{Ts},\mathbf{T}),1} : \mathbf{T} \to \mathbf{Ts}\\
&\mathcal{D}_{(\mathbf{Ts},\mathbf{T}),0}\mathsf{Empty} \mapsto \mathsf{InL}\,\mathsf{U} && \mathcal{D}_{(\mathbf{Ts},\mathbf{T}),1}(\mathsf{Branch}\, as) \mapsto as\\
&\mathcal{D}_{(\mathbf{Ts},\mathbf{T}),0}(\mathsf{Tcons}\, a\, as) \mapsto \mathsf{InR}\,\langle a, as\rangle
\end{aligned}
$$

Example 2.2.33 (Corecursion operators on simultaneous algebras)**.**

$$
\begin{aligned}
&{}^{\mathrm{co}}\mathcal{R}^{(\tau_0,\tau_1)}_{(\mathbf{Ts},\mathbf{T}),0} : \tau_0 \to (\tau_0 \to \mathbf{U} + (\mathbf{T} + \tau_1) \times (\mathbf{Ts} + \tau_0)) \to (\tau_1 \to \mathbf{Ts} + \tau_0) \to \mathbf{Ts}\\
&{}^{\mathrm{co}}\mathcal{R}^{(\tau_0,\tau_1)}_{(\mathbf{Ts},\mathbf{T}),0} \mapsto \lambda_{u,v_0,v_1}(\mathrm{Case}\ v_0 u\ \mathrm{of}\\
&\qquad\qquad \mathsf{InL}\,\mathsf{U} \to \mathsf{Empty}\\
&\qquad\qquad \mathsf{InR}\,x \to \mathsf{Tcons}(\mathcal{M}(\pi_0\,x)[\mathrm{id}, {}^{\mathrm{co}}\mathcal{R}^{\vec{\tau}}_{(\mathbf{Ts},\mathbf{T}),0} \cdot \vec{v}]\,[\mathrm{id}, {}^{\mathrm{co}}\mathcal{R}^{\vec{\tau}}_{(\mathbf{Ts},\mathbf{T}),1} \cdot \vec{v}])\\
&\qquad\qquad\qquad\qquad (\mathcal{M}(\pi_1\,x)[\mathrm{id}, {}^{\mathrm{co}}\mathcal{R}^{\vec{\tau}}_{(\mathbf{Ts},\mathbf{T}),0} \cdot \vec{v}]\,[\mathrm{id}, {}^{\mathrm{co}}\mathcal{R}^{\vec{\tau}}_{(\mathbf{Ts},\mathbf{T}),1} \cdot \vec{v}]))\\
&{}^{\mathrm{co}}\mathcal{R}^{(\tau_0,\tau_1)}_{(\mathbf{Ts},\mathbf{T}),1} : \tau_1 \to (\tau_0 \to \mathbf{U} + (\mathbf{T} + \tau_1) \times (\mathbf{Ts} + \tau_0)) \to (\tau_1 \to \mathbf{Ts} + \tau_0) \to \mathbf{T}\\
&{}^{\mathrm{co}}\mathcal{R}^{(\tau_0,\tau_1)}_{(\mathbf{Ts},\mathbf{T}),1} \mapsto \lambda_{u,v_0,v_1}(\mathsf{Branch}(\mathcal{M}(v_1 u)[\mathrm{id}, {}^{\mathrm{co}}\mathcal{R}^{\vec{\tau}}_{(\mathbf{Ts},\mathbf{T}),m} \cdot \vec{v}]_{m<l}))
\end{aligned}
$$

Example 2.2.34 (Infinite list of ascending natural numbers)**.** By means of corecursion on $\mathbf{L_N}$ we define an infinite list of natural numbers ascending from the given n.

$$
\begin{aligned}
&Asc : \mathbf{N} \to \mathbf{L_N}\\
&Asc := \lambda_n({}^{\mathrm{co}}\mathcal{R}_{\mathbf{N}}\, n\, \lambda_m\langle m, \mathsf{InR}(\mathsf{InR}(\mathsf{S}m))\rangle)
\end{aligned}
$$

The conversion sequence yielded by $Asc\,n$ is

$$Asc\,n \mapsto^* n{::}\,Asc(\mathsf{S}n) \mapsto^* n{::}\mathsf{S}n{::}\,Asc(\mathsf{S}(\mathsf{S}n)) \mapsto^* \ldots.$$

2.2.3 Programmable Constants

Programmable constants are constants defined by specifying a name and its type. One can give computational meaning of such a constant by defining *computation rules* under some conditions. It is also possible to give *rewriting rules* which are more flexible than computation rules. Following Schwichtenberg [SW12], we first define programmable constants and computation rules, then define rewriting rules as well.

Definition 2.2.35 (Programmable constants)**.** A programmable constant is defined by specifying a name C and its type ρ.

We introduce *constructor patterns* in order to formalize computation of programmable constants.

Definition 2.2.36 (Constructor pattern)**.** We inductively define CP.

$$
\frac{}{\mathrm{CP}()}\ ,\qquad \frac{\mathrm{CP}_{\vec{x}}(\vec{P}) \quad \mathrm{CP}_{\vec{y}}(Q) \quad \vec{x},\vec{y}\ \text{are disjoint}}{\mathrm{CP}_{\vec{x},\vec{y}}(\vec{P},Q)}\ ,
$$

$$
\frac{}{\mathrm{CP}_x(x)}\ ,\qquad \frac{\mathrm{C}^{\vec{\tau}\to\iota}\ \text{is a constructor} \quad \mathrm{CP}_{\vec{x}}(\vec{P})}{\mathrm{CP}_{\vec{x}}((\mathrm{C}\vec{P})^{\iota})}\ .
$$

When $\mathrm{CP}_{\vec{x}}(P)$ holds, P is a *constructor pattern* with variables $\vec{x}$.

Definition 2.2.37 (Computation rules)**.** Let D be a programmable constant of type $\vec{\sigma} \to \tau$. Computation rules of D are finitely many equations of the form

$$D\vec{P}_i(\vec{y}_i) = M_i,$$

where free variables of $\vec{P}_i(\vec{y}_i)$ and M are among $\vec{y}_i$ under the following requirements. Each $\vec{P}_i$ is a constructor pattern such that for $i \neq j$, $\vec{P}_i$ and $\vec{P}_j$ have disjoint free variables and are not unifiable. Moreover, the lengths of all $\vec{P}_i(\vec{y}_i)$ are the same. This length is called the arity of D denoted by $\mathrm{ar}(D)$. Each equation is also adopted as a conversion rule from the left to the right.

Example 2.2.38 (Predecessor)**.** Let P be a programmable constant of type $\mathbf{N} \to \mathbf{N}$. The following equations define the predecessor of natural numbers.

$$P\,\mathsf{0} = \mathsf{0}, \qquad P(\mathsf{S}\,n) = n.$$

Even if a rule does not satisfy the condition to be a computation rule, the rule can be defined as a rewriting rule.

Definition 2.2.39 (Rewriting rule)**.** Let D be a programmable constant of type $\vec{\sigma} \to \tau$. Rewriting rules of D are finitely many equations of the form

$$D\vec{N} = M,$$

where N_i is of type σ_i. If $D\vec{N} = M$ is not a proven proposition, the equation has to be added as an axiom. The consistency has to be considered as well. Each equation is also adopted as a conversion rule from the left to the right.

Example 2.2.40 (Addition of natural numbers)**.** Consider the addition on natural numbers by means of a programmable constant. Let $+^{\mathbf{N}\to\mathbf{N}\to\mathbf{N}}$ be a programmable constant. In the infix notation we define the following computation rules.

$$n + \mathsf{0} = n, \qquad n + (\mathsf{S}\,m) = \mathsf{S}(n + m).$$

In addition, we prove the following three equations,

$$\forall_n(\mathsf{0} + n = n), \qquad \forall_{n,m}((\mathsf{S}\,m) + n = \mathsf{S}(n + m)),$$
$$\forall_{n,m,k}(n + (m + k) = (n + m) + k),$$

then add them as rewriting rules.

2.2.4 Denotational Semantics

The term calculus T^+ is a concrete representation of our abstract notion of computability. We describe denotational semantics of T^+ due to Schwichtenberg [SW12] and normalization by evaluation due to Berger and Schwichtenberg [BS91]. Considering ideals as denotation of terms of T^+, we define the relation between a token $(\vec{U}, b)$ and the denotation of a term M, written as $(\vec{U}, b) \in [\![M]\!]$, by induction on the construction of M. We denote $(U_1, \ldots, (U_n, b)\ldots)$ by $(\vec{U}, b)$ and $(\vec{U}, b) \in [\![\lambda_{\vec{x}} M]\!]$ for all $b \in V$ by $(\vec{U}, V) \subseteq [\![\lambda_{\vec{x}} M]\!]$.

Definition 2.2.41 (Denotational semantics)**.** Let b range over tokens, U, V, and W over sets of tokens, x and y over variables, M and N over terms, C over constructors, and D over a constant with computation rules as $D\vec{P}(\vec{y}) = M$ where each of $\vec{P}$ is a constructor pattern. We define the denotation of a term by induction.

$$\frac{U_i \vdash b}{(\vec{U}, b) \in [\![\lambda_{\vec{x}} x_i]\!]}\ (V), \qquad \frac{(\vec{U}, V, b) \in [\![\lambda_{\vec{x}} M]\!] \quad (\vec{U}, V) \subseteq [\![\lambda_{\vec{x}} N]\!]}{(\vec{U}, b) \in [\![\lambda_{\vec{x}}(MN)]\!]}\ (A),$$

$$\frac{\vec{V} \vdash \vec{b}^*}{(\vec{U}, \vec{V}, C\vec{b}^*) \in [\![\lambda_{\vec{x}} C]\!]}\ (C), \qquad \frac{(\vec{U}, \vec{V}, b) \in [\![\lambda_{\vec{x},\vec{y}} M]\!] \quad \vec{W} \vdash \vec{P}(\vec{V})}{(\vec{U}, \vec{W}, b) \in [\![\lambda_{\vec{x}} D]\!]}\ (D).$$

The denotation of a term in T^+ is an ideal as expected. Essential properties as the preservation of values and adequacy with respect to operational semantics based on conversion rules also enjoy. See [BS91, BES03] for details.

One application of the denotational semantics is *normalization by evaluation*, NbE in short, which is an efficient algorithm to normalize terms [BS91, BES03, SW12]. It is also a theoretical basis of the implementation of normalization in the Minlog system. A term is a normal form if no conversion rule applies. For a given term M such a long normal form, written $\mathrm{nf}(M)$, is computed by NbE in the following way: We first evaluate M to an object a in some denotational semantics. Then, we retrieve from a the long normal form $\mathrm{nf}(M)$. We define the interpretation of types as follows.

Definition 2.2.42 (Interpretation of types)**.** Let Λ_ρ be the set of all terms of type ρ. We define the interpretation of types by induction on a type.

$$[\![\iota]\!] \coloneqq \Lambda_\iota, \qquad [\![\rho \to \sigma]\!] \coloneqq [\![\sigma]\!]^{[\![\rho]\!]}$$

Meaning of free variables in an object can be given by an assignment $\uparrow$ lifting a variable to an object. We write $[\![M]\!]_\uparrow$ to mean the object of M under the assignment $\uparrow$. We also consider a function $\downarrow$ which retrieves a long normal form from an object. We simultaneously define $\uparrow$ and $\downarrow$, called reflect and reify, respectively. For convenience we define $\uparrow$ on terms rather than on variables.

Definition 2.2.43 (Reflect and reify)**.** We simultaneously define two functions $\uparrow_\rho \colon \Lambda_\rho \to [\![\rho]\!]$ and $\downarrow_\rho \colon [\![\rho]\!] \to \Lambda_\rho$.

$$\begin{aligned} &\uparrow_\iota (M) \coloneqq M, && \downarrow_\iota (M) \coloneqq M, \\ &\uparrow_{\rho\to\sigma} (M)(a) \coloneqq \uparrow_\sigma (M \downarrow_\rho (a)), && \downarrow_{\rho\to\sigma} (a) \coloneqq \lambda_x(\downarrow_\sigma (a(\uparrow_\rho (x)))), \end{aligned}$$

where x is a fresh variable.

A problem is that what x is in the above definition. Based on the idea of de Bruijn's index, we introduce *term families* due to Berger and Schwichtenberg [BS91] and Filinski [Fil99] to avoid this difficulty.

Definition 2.2.44 (Term family)**.** To a term M^ρ we assign a *term family* $M^\infty : \mathbf{N} \to \Lambda_\rho$ by induction on the construction of M.

$$x^\infty(k) \coloneqq x, \qquad (\lambda_y M)^\infty(k) \coloneqq \lambda_{x_k}(M[y/x_k]^\infty(k+1)),$$
$$c^\infty(k) \coloneqq c, \qquad (MN)^\infty(k) \coloneqq M^\infty(k)N^\infty(k),$$

where c is a constant.

By using term families, we can formalize Definition 2.2.43 without using a variable name.

For term families $r : \mathbf{N} \to \Lambda_{\rho\to\sigma}$ and $s : \mathbf{N} \to \Lambda_\rho$, the application rs is defined by $rs(k) \coloneqq r(k)s(k)$. The following is the standard way to get a term from a term family.

Definition 2.2.45 (Extraction from a term family)**.** Let r be a term family and k be the greatest index among indices of free and bound variables x_i which occur in $r(0)$ We define $\mathrm{ext}(r)$ by the term $r(k+1)$.

Now we can give a refined version of NbE which is suitable for the implementation.

Definition 2.2.46 (Refined interpretation of types)**.** Let Λ_ρ be the set of all terms of type ρ and $\Lambda_\rho^{\mathbf{N}}$ be the function space from natural numbers to Λ_ρ. We define the refined interpretation of types by induction on a type.

$$[\![\iota]\!] \coloneqq \Lambda_\iota^{\mathbf{N}}, \qquad [\![\rho \to \sigma]\!] \coloneqq [\![\sigma]\!]^{[\![\rho]\!]},$$

Definition 2.2.47 (Refined reflect and reify)**.** We simultaneously define two functions $\uparrow_\rho : (\mathbf{N} \to \Lambda_\rho) \to [\![\rho]\!]$ and $\downarrow_\rho : [\![\rho]\!] \to \mathbf{N} \to \Lambda_\rho$.

$$\uparrow_\iota (r) \coloneqq r, \qquad \downarrow_\iota (r) \coloneqq r,$$
$$\uparrow_{\rho\to\sigma} (r)(a) \coloneqq \uparrow_\sigma (r \downarrow_\rho (a)), \qquad \downarrow_{\rho\to\sigma} (a)(k) \coloneqq \lambda^\rho_{x_k}(\downarrow_\sigma (a(\uparrow_\rho (x_k^\infty)))(k+1)).$$

For $a_i \in [\![\rho_i]\!]$, $\uparrow_{\vec{\rho}\to\sigma} (r)(a_1, \ldots, a_n) \coloneqq \uparrow_\sigma (r \downarrow_{\rho_1} (a_1) \ldots \downarrow_{\rho_n} (a_n))$ holds. The correctness of NbE is shown as follows.

Theorem 2.2.48 (Correctness of NbE)**.** *Let M be a term in $\beta\eta$-long normal form. We have $\downarrow ([\![M]\!]_\uparrow) = M^\infty$ where $[\![M]\!]_\uparrow$ denotes the (semantic) object of M under the assignment given by $\uparrow$.*

Proof. By induction on the construction of M. □

We describe the implementation of NbE in Section A.1.1.

2.3 Inductive and Coinductive Definitions

Inductive and coinductive definitions are means to define predicates inductively and coinductively, which makes TCF extensible. These definitions provide a way to specify a prime formula, which is sometimes not concretely given, for example in the Brouwer-Heyting-Kolmogorov interpretation of intuitionistic logic.

We give the notion of formulas and predicates simultaneously.

Definition 2.3.1 (Formula forms and predicate forms)**.** We simultaneously define formula forms A, B and predicate forms P. Let X denote a predicate variable with an arity, and the length of $\vec{t}$ be the same as the length of the arity of P.

$$\begin{aligned}&A, B ::= P\vec{t} \mid A \to B \mid \forall_x A,\\ &P ::= X \mid \{\vec{x} \mid A\} \mid \mu_X(\forall_{\vec{x}_i}((A_{i\nu})_{\nu<n_i} \to X\vec{t}_i))_{i<k} \mid \nu_X(\forall_{\vec{x}_i}((A_{i\nu})_{\nu<n_i} \to X\vec{t}_i))_{i<k}.\end{aligned}$$

We abbreviate $A_0 \to \ldots \to A_{n-1} \to B$ as $(A_i)_{i<n} \to B$ and $\forall_{x_0} \ldots \forall_{x_{n-1}} A$ as $\forall_{\vec{x}} A$.

Definition 2.3.2 (Free predicate variables)**.** We simultaneously define free predicate variables in a formula form and a predicate form.

$$\begin{aligned}&\mathrm{FPV}(P\vec{t}) \coloneqq \mathrm{FPV}(P),\\ &\mathrm{FPV}(A \to B) \coloneqq \mathrm{FPV}(A) \cup \mathrm{FPV}(B),\\ &\mathrm{FPV}(\forall_x A) \coloneqq \mathrm{FPV}(A),\\ &\mathrm{FPV}(X) \coloneqq \{X\},\\ &\mathrm{FPV}(\{\vec{x} \mid A\}) \coloneqq \mathrm{FPV}(A),\\ &\mathrm{FPV}(\mu_X(\forall_{\vec{x}_i}((A_{i\nu})_{\nu<n_i} \to X\vec{t}_i))_{i<k}) \coloneqq \textstyle\bigcup_{i<k} \bigcup_{\nu<n_i} \mathrm{FPV}(A_{i\nu}) \setminus \{X\},\\ &\mathrm{FPV}(\nu_X(\forall_{\vec{x}_i}((A_{i\nu})_{\nu<n_i} \to X\vec{t}_i))_{i<k}) \coloneqq \textstyle\bigcup_{i<k} \bigcup_{\nu<n_i} \mathrm{FPV}(A_{i\nu}) \setminus \{X\}.\end{aligned}$$

We can denote $A(X)$ instead of A to express that $A(Y)$ means replacing X in A by Y.

Definition 2.3.3 (Strictly positive occurrence of predicate variables)**.** We define $\mathrm{SP}_{\vec{Y}}$ on a formula form and a predicate form.

$$\frac{\mathrm{SP}_{\vec{Y}}(P)}{\mathrm{SP}_{\vec{Y}}(P\vec{t})}\ , \quad \frac{\mathrm{FPV}(A) \cap \vec{Y} = \emptyset \quad \mathrm{SP}_{\vec{Y}}(B)}{\mathrm{SP}_{\vec{Y}}(A \to B)}\ , \quad \frac{\mathrm{SP}_{\vec{Y}}(A)}{\mathrm{SP}_{\vec{Y}}(\forall_x A)}\ , \quad \frac{}{\mathrm{SP}_{\vec{Y}}(X)}\ , \quad \frac{\mathrm{SP}_{\vec{Y}}(A)}{\mathrm{SP}_{\vec{Y}}(\{\vec{x} \mid A\})}\ ,$$
$$\frac{\text{for all } i<k, \text{ for all } \nu<n_i,\ \mathrm{SP}_{\vec{Y},X}(A_{i\nu})}{\mathrm{SP}_{\vec{Y}}(\mu_X(\forall_{\vec{x}_i}((A_{i\nu})_{\nu<n_i} \to X\vec{t}_i))_{i<k})}\ , \quad \frac{\text{for all } i<k, \text{ for all } \nu<n_i,\ \mathrm{SP}_{\vec{Y},X}(A_{i\nu})}{\mathrm{SP}_{\vec{Y}}(\nu_X(\forall_{\vec{x}_i}((A_{i\nu})_{\nu<n_i} \to X\vec{t}_i))_{i<k})}\ .$$

Occurrences of $\vec{Y}$ in A (P) is *strictly positive* if $\mathrm{SP}_{\vec{Y}}(A)$ ($\mathrm{SP}_{\vec{Y}}(P)$).

Definition 2.3.4 (Inhabited formula forms and predicate forms)**.** Let variables X and Y range over predicate forms. We simultaneously define $\mathrm{Inhab}_{\vec{Y}}$ on a formula form and a

predicate form as follows.

$$\frac{\mathrm{Inhab}_{\vec{Y}}(X)}{\mathrm{Inhab}_{\vec{Y}}(X\vec{t})}\ ,\quad \frac{\mathrm{Inhab}_{\vec{Y}}(B)}{\mathrm{Inhab}_{\vec{Y}}(A\to B)}\ ,\quad \frac{\mathrm{Inhab}_{\vec{Y}}(A)}{\mathrm{Inhab}_{\vec{Y}}(\forall_{\vec{x}}A)}\ ,\quad \frac{X\notin\vec{Y}}{\mathrm{Inhab}_{\vec{Y}}(X)}\ ,\quad \frac{\mathrm{Inhab}_{\vec{Y}}(A)}{\mathrm{Inhab}_{\vec{Y}}(\{\vec{x}\,|\,A\})}\ ,$$
$$\frac{\text{there exists } i<k, \text{ for all } \nu<n_i,\ \mathrm{Inhab}_{\vec{Y},X}(A_{i\nu})}{\mathrm{Inhab}_{\vec{Y}}(\mu_X(\forall_{\vec{x}_i}((A_{i\nu})_{\nu<n_i}\to X\vec{t}_i))_{i<k}))}\ ,$$
$$\frac{\text{there exists } i<k, \text{ for all } \nu<n_i,\ \mathrm{Inhab}_{\vec{Y},X}(A_{i\nu})}{\mathrm{Inhab}_{\vec{Y}}(\nu_X(\forall_{\vec{x}_i}((A_{i\nu})_{\nu<n_i}\to X\vec{t}_i))_{i<k}))}\ .$$

A formula form A is *inhabited* if $\mathrm{Inhab}(A)$ holds, where the subscript is omitted to mean that the empty list of predicate variables is given. A formula form A is *absolutely inhabited* if $\mathrm{Inhab}_{\mathrm{FPV}(A)}(A)$.

Definition 2.3.5 (Formulas and predicates)**.** We simultaneously define F and Pred.

$$\frac{\mathrm{Pred}(P)}{\mathrm{F}(P\vec{t})}\ ,\quad \frac{\mathrm{F}(A)\quad \mathrm{F}(B)}{\mathrm{F}(A\to B)}\ ,\quad \frac{\mathrm{F}(A)}{\mathrm{F}(\forall_x A)}\ ,\quad \frac{}{\mathrm{Pred}(X)}\ ,\quad \frac{\mathrm{F}(A)}{\mathrm{Pred}(\{\vec{x}\,|\,A\})}\ ,$$
$$\frac{\text{for all } i<k,\ \mathrm{F}(X\vec{t}_i) \text{ and for all } \nu<n_i,\ \mathrm{F}(A_{i\nu}) \text{ and } \mathrm{SP}_X(A_{i\nu})\quad \mathrm{Inhab}(I)}{\mathrm{Pred}(I)}\ ,$$
$$\frac{\text{for all } i<k,\ \mathrm{F}(X\vec{t}_i) \text{ and for all } \nu<n_i,\ \mathrm{F}(A_{i\nu}) \text{ and } \mathrm{SP}_X(A_{i\nu})\quad \mathrm{Inhab}({}^{\mathrm{co}}I)}{\mathrm{Pred}({}^{\mathrm{co}}I)}\ ,$$

where $I \coloneqq \mu_X(\forall_{\vec{x}_i}((A_{i\nu})_{\nu<n_i}\to X\vec{t}_i))_{i<k}$ and ${}^{\mathrm{co}}I \coloneqq \nu_X(\forall_{\vec{x}_i}((A_{i\nu})_{\nu<n_i}\to X\vec{t}_i))_{i<k}$. A formula form A is a *formula* if $\mathrm{F}(A)$, and a predicate form P is a *predicate* if $\mathrm{Pred}(P)$.

Inductively defined predicate constants, also called inductive predicates in short, extend the system by adding the introduction and the elimination axioms. Similarly in the case of coinductively defined predicate constants, definitions extend the system by adding the closure and the greatest-fixed-point axioms.

2.3.1 Inductive Definitions

We give the definition of inductive predicates and its axioms.

Definition 2.3.6 (Inductively defined predicates)**.** Suppose that formulas K_i for $i<k$ are of the form $\forall_{\vec{x}_i}((A_{i\nu})_{\nu<n_i}\to X\vec{t}_i)$, then $I := \mu_X(\vec{K})$ is an *inductively defined predicate* if $\mathrm{Pred}(I)$. We denote $A_{i\nu}(X)$ to describe the free occurrences of X in $A_{i\nu}$. The introduction axioms are obtained by replacing each occurrence of X in $\vec{K}$ by I.

$$\forall_{\vec{x}}((A_{i\nu}(I))_{\nu<n_i}\to I\vec{t}). \qquad I_i^+$$

We refer to each introduction axiom of I derived from K_i by I_i^+, possibly with additional parentheses for the readability. Let P be a predicate variable of the same arity as I. The elimination axiom of I is of the following form,

$$\forall_{\vec{x}}(I\vec{x} \to (B_i(I,P))_{i<k} \to P\vec{x}), \qquad I^-$$

which is referred to by I^-. Let $I \cap P$ be $\{\vec{x} \mid I\vec{x} \land P\vec{x}\}$, then $B_i(I,P)$ is given as follows,

$$\forall_{\vec{x}}((A_{i\nu}(I \cap P))_{\nu<n_i} \to P\vec{t}),$$

namely, by replacing the final occurrence of X in K_i by P and all other X by $I \cap P$. Conventionally, when an occurrence of X in the premise of the implication in K_i is not given as a parameter argument of another predicate constant, we duplicate I and P by currying rather than using conjunction. We call $A_{i\nu}(X)$ a parameter premise if it does *not* contain X, otherwise a recursive premise. Occurrences of X as parameter arguments of a previously defined predicate are called *nested*, and predicates involving a nested occurrence of X are called *nested inductively defined predicates*.

Roughly speaking, the elimination axiom I^- states that if a predicate P can replace I in I_i^+ for all $i < k$, I is smaller than P. In our formulation, the strengthening is used in the step formulas, so that $I \cap P$ is assumed in premises instead of P. When there is no confusion, we can give a list of introduction axioms instead of $\mu_X(\vec{K})$ in order to define an inductive predicate. To make parameter predicates explicit, we write parameters as subscripts, as $I_{\vec{X}}$ where $\vec{X}$ are parameter predicates of I. For readability, it can be written by parentheses as well, i.e., $I(\vec{X})$.

We take a look at one example of inductive definition.

Example 2.3.7 (Leibniz equality)**.** We define Leibniz equality by

$$\mathrm{eqd}^\rho := \mu_X(\forall_{x^\rho} X(x,x)),$$

where ρ is a type parameter. The introduction and elimination axioms are

$$\forall_{x^\rho}(\mathrm{eqd}(x,x)), \qquad \mathrm{eqd}^+$$
$$\forall_{x,y}(\mathrm{eqd}(x,y) \to \forall_x Pxx \to Pxy), \qquad \mathrm{eqd}^-$$

where $\mathrm{eqd}(x,y)$ abbreviates $\mathrm{eqd}^\rho(x^\rho, y^\rho)$. We adopt the infix notation as $x\,\mathrm{eqd}\,y$.

Lemma 2.3.8 (Compatibility of eqd)**.** $\forall_{x,y}(x \text{ eqd } y \to Px \to Py)$.

Proof. Let x and y be given and assume x eqd y. We use eqd^- for x eqd y to prove the goal $Px \to Py$. The step formula $\forall_x(Px \to Px)$ is trivial. □

We define truth $\mathbf{T}$ and falsity $\mathbf{F}$ to be T eqd T and $\mathsf{F}\,\mathrm{eqd}\,\mathsf{T}$, respectively. We have *ex-falso-quodlibet* for a certain class of formulas in minimal logic. The proof deals with the case of coinductively defined predicates which will be described in Section 2.3.2.

Theorem 2.3.9 (Ex-falso-quodlibet)**.** *Let A be an absolutely inhabited formula, then there is a proof of $\mathbf{F} \to A$.*

Proof. First, we claim that arbitrary x^ρ and y^ρ are Leibniz equal if $\mathbf{F}$ is assumed. Let x^ρ and y^ρ be given, and assume $\mathbf{F}$. Trivially, $(\mathcal{R}^\rho_{\mathbf{B}}\,\mathsf{F}\,x\,y)\,\mathrm{eqd}\,(\mathcal{R}^\rho_{\mathbf{B}}\,\mathsf{T}\,y\,x)$ comes by eqd^+. We use 2.3.8 with $\mathsf{F}\,\mathrm{eqd}\,\mathsf{T}$, then $(\mathcal{R}^\rho_{\mathbf{B}}\,\mathsf{T}\,x\,y)\,\mathrm{eqd}\,(\mathcal{R}^\rho_{\mathbf{B}}\,\mathsf{T}\,y\,x)$, namely, $x\,\mathrm{eqd}\,y$ also holds.

Let $\vec{X}$ be free predicate variables in A. By induction on the construction of A. *Case* $A_0 \to A_1$. Assume $\mathbf{F}$ and A_0, then use induction hypothesis for A_1. *Case* $\forall_{\vec{x}} A_0$. Assume $\mathbf{F}$ and let $\vec{x}$ be given, then use induction hypothesis for A_0. *Case* $I\vec{t}$. We have the derivation of $\mathrm{Inhab}_{\vec{X}}(I\vec{t})$, where an index i is determined in the application of the last rule in Definition 2.3.4. Using the claim at the beginning, our goal is same as $I\vec{s}$, so that the I_i^+ applies, then all premises of I_i^+ come by induction hypothesis. *Case* ${}^{\mathrm{co}}I\vec{t}$. We use ${}^{\mathrm{co}}I^+$ with a competitor predicate $P := \{\vec{x} \mid \mathbf{F}\}$. The first premise is trivial. We prove the second premise $\forall_{\vec{y}}(P\vec{y} \to \bigvee_{i<k} \exists_{\vec{x}_i}(\bigwedge_{\nu<n_i} A_{i\nu}({}^{\mathrm{co}}I \cup P) \land \bigwedge \vec{E}_i))$. Let $\vec{y}$ be given and assume $P\vec{y}$. We have the derivation of $\mathrm{Inhab}_{\vec{X}}({}^{\mathrm{co}}I\vec{t})$, where an index i is determined in the application of the last rule in Definition 2.3.4. Using $\bigvee^+$, it suffices to prove $\exists_{\vec{x}_i}(\bigwedge_{\nu<n_i} A_{i\nu}({}^{\mathrm{co}}I \cup P) \land \bigwedge \vec{E}_i)$ which follows from induction hypothesis. □

We can define conjunction, disjunction and existential quantifier in our setting, following Martin-Löf [ML71].

Example 2.3.10 (Conjunction, disjunction and existential quantifier)**.** Conjunction, disjunction and the existential quantifier are inductively defined as predicates. Let A and B be propositional variables, i.e., predicate variables of arity 0, and C be a predicate variable of arity (α). We inductively define Conj, Disj and Ex.

$$\mathrm{Conj}_{A,B} := \mu_X(A \to B \to X), \qquad \mathrm{Disj}_{A,B} := \mu_X(A \to X, B \to X)$$
$$\mathrm{Ex}_C := \mu_X(\forall_{x^\alpha}(C\,x \to X)).$$

We can write $A \land B$, $A \lor B$ and $\exists_x A$ instead of $\mathrm{Conj}_{\{|A\},\{|B\}}$, $\mathrm{Disj}_{\{|A\},\{|B\}}$ and $\mathrm{Ex}_{\{x\,|\,A\}}$. The introduction axioms are as follows.

$$A \to B \to A \land B, \qquad \land^+$$
$$A \to A \lor B, \qquad \lor_0^+$$
$$B \to A \lor B, \qquad \lor_1^+$$
$$\forall_{x^\alpha}(A \to \exists_x A), \qquad \exists^+$$

Let P be a propositional variable. The elimination axioms are given as follows.

$$A \land B \to (A \to B \to P) \to P, \qquad \land^-$$
$$A \lor B \to (A \to P) \to (B \to P) \to P, \qquad \lor^-$$
$$\exists_x A \to \forall_x(A \to P) \to P. \qquad \exists^-$$

Notice that we need e.g. conjunction in the definition of axioms of inductive predicates with a recursive argument, but the definitions of conjunction, disjunction and existential quantifier are recursive arguments free.

Example 2.3.11 (Even numbers)**.** Consider a predicate Even of arity $(\mathbf{N})$. We define Even to be $\mu_X(K_0, K_1)$ such that

$$K_0 = X\,\mathsf{0}, \qquad K_1 = \forall_n(X\,n \to X(\mathsf{S}(\mathsf{S}n))).$$

The introduction axioms are

$$\begin{aligned} &\mathrm{Even}\,\mathsf{0}, && \mathrm{Even}_0^+\\ &\forall_n(\mathrm{Even}\,n \to \mathrm{Even}(\mathsf{S}(\mathsf{S}n))). && \mathrm{Even}_1^+ \end{aligned}$$

Let P be a predicate variable of arity $(\mathbf{N})$, then the elimination axiom is

$$\forall_n(\mathrm{Even}\,n \to P\,\mathsf{0} \to \forall_n(\mathrm{Even}\,n \to P\,n \to P(\mathsf{S}(\mathsf{S}\,n))) \to P\,n). \qquad \mathrm{Even}^-$$

Example 2.3.12 (Pointwise equality)**.** Apart from the Leibniz equality, the pointwise equality is inductively definable as well. On $\mathbf{N}$, we define the pointwise equality $=_{\mathbf{N}}$ of arity $(\mathbf{N}, \mathbf{N})$ as follows:

$$=_{\mathbf{N}} := \mu_X(X\mathsf{00}, \forall_{n,m}(Xnm \to X(\mathsf{S}n)(\mathsf{S}m))).$$

The introduction and elimination axioms are

$$\begin{aligned} &\mathsf{0} =_{\mathbf{N}} \mathsf{0}, && (=_{\mathbf{N}})_0^+\\ &\forall_{n,m}(n =_{\mathbf{N}} m \to \mathsf{S}\,n =_{\mathbf{N}} \mathsf{S}\,m), && (=_{\mathbf{N}})_1^+\\ &\forall_{n,m}(n =_{\mathbf{N}} m \to P\mathsf{00} \to \forall_{n,m}(n =_{\mathbf{N}} m \to Pnm \to P(\mathsf{S}\,n, \mathsf{S}\,m)) \to Pnm). && (=_{\mathbf{N}})^- \end{aligned}$$

For any finitary algebra ι, we can also define the pointwise equality on $\mathbf{L}_\iota$ which is relativised by a relation R_ι on ι. We define $=_{\mathbf{L}_\iota}$ to be

$$\mu_X(X[][], \forall_{x,y}(R_\iota xy \to \forall_{xs,ys}(Xxsys \to X(x::xs)(y::ys)))).$$

The introduction and elimination axioms are

$$\begin{aligned} &[] =_{\mathbf{L}_\iota} [], && (=_{\mathbf{L}_\iota})_0^+\\ &\forall_{x,y}(R_\iota xy \to \forall_{xs,ys}(xs =_{\mathbf{L}_\iota} ys \to x::xs =_{\mathbf{L}_\iota} y::ys), && (=_{\mathbf{L}_\iota})_1^+\\ &\forall_{xs,ys}(xs =_{\mathbf{L}} ys \to P[][] \to \\ &\qquad \forall_{x,y}(R_\iota xy \to \forall_{xs,ys}(xs =_{\mathbf{L}} ys \to Pxsys \to P(x::xs, y::ys))) \to && (=_{\mathbf{L}_\iota})^-\\ &\qquad Pxsys). \end{aligned}$$

See also Example 2.3.21 for nested definitions.

2.3.2 Coinductive Definitions

Coinductively defined predicates come as companion predicates of inductively defined ones. We generalize conjunction and disjunction to formulate coinductive predicates.

Definition 2.3.13 (Generalized conjunction)**.** Assume a list of formulas $\vec{A}$ of length $n \geq 0$. We define generalized conjunction $\bigwedge \vec{A}$ by induction on the length of $\vec{A}$.

$$\bigwedge() \coloneqq \mathbf{T}, \quad \bigwedge(A) \coloneqq A, \quad \bigwedge(A, \vec{A}) \coloneqq A \wedge \bigwedge \vec{A}.$$

Let n be the length of $\vec{A}$. We define $\bigwedge_{i<n} A_i$ to be $\bigwedge \vec{A}$.

Instead of repeated application of $\wedge^+$, we can call it $\bigwedge^+$. In the same way, instead of repeated application of $\wedge^-$, we can say $\bigwedge^-$.

Definition 2.3.14 (Generalized disjunction)**.** Assume propositions $\vec{A}$ of length $n \geq 0$. We define disjunction $\bigvee \vec{A}$ by induction on the length of $\vec{A}$.

$$\bigvee() \coloneqq \mathbf{F}, \quad \bigvee(A) \coloneqq I_A, \quad \bigvee(A, \vec{A}) \coloneqq \tilde{\bigvee}(A, \vec{A})$$
$$\tilde{\bigvee}(A) \coloneqq A, \quad \tilde{\bigvee}(A, \vec{A}) \coloneqq A \vee \tilde{\bigvee}(\vec{A}),$$

where $I_Y \coloneqq \mu_X(Y \to X)$. We define $\bigvee_{i<n} \vec{A}$ to be $\bigvee \vec{A}$.

There is a special treatment for the case of a singleton list. This is a way to manage the typing issue of a closure axiom of a one-clause coinductive definition without computational assumptions, for example the cototality predicate of $\mathbf{U}$, in Section 2.5.1. For n, n' such that $n \leq n' \leq \nu$, we also allow to write $\bigvee_{i;n\leq i<n'}(A_i)_{i<\nu}$ to mean $\bigvee(A_{i+n})_{i<\nu-n'}$, which is a sublist of $(A_i)_{i<\nu}$ defined to be

$$(A_{i+n})_{i<0} \coloneqq (), \qquad (A_{i+n})_{i<m+1} \coloneqq (A_n, (A_{i+n+1})_{i<m}).$$

Instead of repeating $\vee_0^+$ and $\vee_1^+$, we can say $\bigvee^+$ if there is no confusion. In the same way, instead of repeated application of $\vee^-$, we can say $\bigvee^-$. We abbreviate $\bigwedge_{i<n} A_i$ and $\bigvee_{i<n} A_i$ as $\bigwedge_i A_i$ and $\bigvee_i A_i$, respectively, when the length of $\vec{A}$ is not essential or there is no confusion.

Definition 2.3.15 (Coinductively defined predicates)**.** Suppose that for each $i < k$ a formula K_i of the form $\forall_{\vec{x}_i}((A_{i\nu}(X))_{\nu<n_i} \to X\vec{t}_i)$ is given. Then, ${}^{\mathrm{co}}I \coloneqq \nu_X(\vec{K})$ is a *coinductively defined predicate* if $\mathrm{Pred}({}^{\mathrm{co}}I)$. The closure axiom ${}^{\mathrm{co}}I^-$ is given as follows. Assume fresh variables $\vec{y}$ with the same length and types of $\vec{t}_i$ and define formulas $E_j \coloneqq y_j \operatorname{eqd} t_j$ for $0 \leq j < |\vec{t}|$. Then ${}^{\mathrm{co}}I^-$ is defined to be

$$\forall_{\vec{y}}({}^{\mathrm{co}}I\vec{y} \to \bigvee_{i<k} \exists_{\vec{x}}(\bigwedge_{\nu<n_i} A_{i\nu}({}^{\mathrm{co}}I) \wedge \bigwedge \vec{E}_i)). \tag{${}^{\mathrm{co}}I^-$}$$

Assume a predicate variable P of the same arity as ${}^{\mathrm{co}}I$. The greatest-fixed-point axiom ${}^{\mathrm{co}}I^+$ is defined to be

$$\forall_{\vec{y}}(P\vec{y} \to \forall_{\vec{y}}(P\vec{y} \to \bigvee_{i<k} \exists_{\vec{x}}(\bigwedge_{\nu<n_i} A_{i\nu}({}^{\mathrm{co}}I \cup P) \wedge \bigwedge \vec{E}_i)) \to {}^{\mathrm{co}}I\vec{y}). \tag{${}^{\mathrm{co}}I^+$}$$

where ${}^{\mathrm{co}}I \cup P$ is $\{\vec{z} \mid {}^{\mathrm{co}}I\vec{z} \vee P\vec{z}\}$. We call the second premise the *costep formula*.

Roughly speaking, the greatest fixed point axiom ${}^{\mathrm{co}}I^{+}$ states that ${}^{\mathrm{co}}I$ is greater than an arbitrary predicate P if P can replace ${}^{\mathrm{co}}I$ in ${}^{\mathrm{co}}I^{-}$. In our formulation, the strengthening is used in the costep formula, so that ${}^{\mathrm{co}}I \cup P$ comes in the conclusion of a costep formula instead of P.

Example 2.3.16 (Coeven numbers)**.** We define ${}^{\mathrm{co}}\mathrm{Even}$, the companion predicate of Even, as well. The closure axiom is

$$\forall_n({}^{\mathrm{co}}\mathrm{Even}\, n \to n \,\mathrm{eqd}\, \mathsf{0} \vee \exists_m({}^{\mathrm{co}}\mathrm{Even}\, m \wedge n \,\mathrm{eqd}\, \mathsf{S}(\mathsf{S}\, m))), \qquad {}^{\mathrm{co}}\mathrm{Even}^{-}$$

and the greatest-fixed-point axiom is

$$\forall_n(P\, n \to \forall_n(P\, n \to n \,\mathrm{eqd}\, \mathsf{0} \vee \exists_m(({}^{\mathrm{co}}\mathrm{Even}\, m \vee P\, m) \wedge n \,\mathrm{eqd}\, \mathsf{S}(\mathsf{S}\, m))) \to {}^{\mathrm{co}}\mathrm{Even}\, n). \qquad {}^{\mathrm{co}}\mathrm{Even}^{+}$$

Example 2.3.17 (Bisimilarity)**.** We consider a companion predicate of $=_{\mathbf{L}_\iota}$, written $\approx$ instead of ${}^{\mathrm{co}}=_{\mathbf{L}_\iota}$. Let R be a relation on ι, then $\approx$ is defined by

$$\begin{aligned}
&\forall_{xs,ys}(xs \approx ys \to (xs \,\mathrm{eqd}\, [] \wedge ys \,\mathrm{eqd}\, [])\vee \\
&\qquad \exists^{\mathrm{r}}_{x',y',xs',ys'}(Rx'y' \wedge xs' \approx ys' \wedge xs \,\mathrm{eqd}\, x'::xs' \wedge ys \,\mathrm{eqd}\, y'::ys')) && \approx^{-}\\
&\forall_{xs,ys}(P\, xs\, ys \to \\
&\quad \forall_{xs,ys}(P\, xs\, ys \to (xs \,\mathrm{eqd}\, [] \wedge ys \,\mathrm{eqd}\, [])\vee \\
&\qquad \exists^{\mathrm{r}}_{x',y',xs',ys'}(Rx'y' \wedge (xs' \approx ys' \vee P\, xs'\, ys') \wedge && \approx^{+}\\
&\qquad\quad xs \,\mathrm{eqd}\, x'::xs' \wedge ys \,\mathrm{eqd}\, y'::ys')) \to \\
&\quad xs \approx ys)
\end{aligned}$$

We call it the *bisimilarity relation* relativised by R.

Example 2.3.18 (Bisimilarity between infinite lists)**.** As a concrete example of the use of greatest-fixed-point axioms, we prove the bisimilarity of two differently defined infinite lists of even numbers. Here we use $\approx$ relativised by eqd on natural numbers. Define the following two terms.

$$\begin{aligned}
l_0 &= {}^{\mathrm{co}}\mathcal{R}^{\mathbf{N}}_{\mathbf{L_N}}\mathsf{0}M_0, \text{ where } M_0 := \lambda_n \mathsf{InR}\langle n, \mathsf{InR}(n+2)\rangle,\\
l_1 &= {}^{\mathrm{co}}\mathcal{R}^{\mathbf{N}}_{\mathbf{L_N}}\mathsf{0}M_1, \text{ where } M_1 := \lambda_n \mathsf{InR}\langle 2n, \mathsf{InR}(n+1)\rangle.
\end{aligned}$$

Both of them unfold into a list $0::2::4::\ldots$ We prove the bisimilarity of them, i.e., $l_0 \approx l_1$, by means of $\approx^{+}$ with the competitor predicate

$$Q := \{s, t \mid \exists_n(s \,\mathrm{eqd}\, {}^{\mathrm{co}}\mathcal{R}^{\mathbf{N}}_{\mathbf{L_N}}\, 2n\, M_0 \wedge t \,\mathrm{eqd}\, {}^{\mathrm{co}}\mathcal{R}^{\mathbf{N}}_{\mathbf{L_N}}\, n\, M_1)\}.$$

It suffices to prove $Q\, l_0\, l_1$ and the costep formula. For the latter, let l_0' and l_1' be given and assume $Q\, l_0'\, l_1'$, namely, there is n' such that $l_0' \,\mathrm{eqd}\, {}^{\mathrm{co}}\mathcal{R}^{\mathbf{N}}_{\mathbf{L_N}}\, 2n'\, M_0$ and $l_1' \,\mathrm{eqd}\, {}^{\mathrm{co}}\mathcal{R}^{\mathbf{N}}_{\mathbf{L_N}}\, n'\, M_1$. We prove the right disjunct of the costep formula. By unfolding, $l_0' = 2n'::{}^{\mathrm{co}}\mathcal{R}^{\mathbf{N}}_{\mathbf{L_N}}(2n'+2)M_0$ and $l_1' = 2n'::{}^{\mathrm{co}}\mathcal{R}^{\mathbf{N}}_{\mathbf{L_N}}(n'+1)M_1$. Since $2n'+2 \,\mathrm{eqd}\, 2(n'+1)$, $Q({}^{\mathrm{co}}\mathcal{R}^{\mathbf{N}}_{\mathbf{L_N}}(2n'+2)M_0, {}^{\mathrm{co}}\mathcal{R}^{\mathbf{N}}_{\mathbf{L_N}}(n'+1)M_1)$ holds.

2.3.3 Totality and Cototality

We have taken a look at *totality* and *cototality* of finitary algebras in Section 2.1. We state them in general for an arbitrary type by inductive and coinductive definitions. We define the totality predicate and the cototality predicate of an arbitrary type.

Definition 2.3.19 (Totality and cototality predicates)**.** Assume τ is a type and θ is a mapping from each free variable α in τ into a predicate variable of arity (α). We define a translation tl from a type τ into totality and cototality predicates of τ.

$$\begin{aligned}
&\mathrm{tl}_\theta(\alpha) \coloneqq \theta(\alpha) \quad \text{if } \alpha \text{ is a variable,}\\
&\mathrm{tl}_\theta(\sigma \to \tau) \coloneqq \{y^{\sigma\to\tau} \mid \forall_{z^\sigma}(\mathrm{tl}_\theta(\sigma)(z) \to \mathrm{tl}_\theta(\tau)(yz))\} \quad y \text{ and } z \text{ are fresh,}\\
&\mathrm{tl}_\theta(\mu_\xi((\rho_{i\nu})_{\nu<n_i} \to \xi)_{i<k}) \coloneqq \hat{\mu}_X(\forall_{\vec{x}_i}((\mathrm{tl}_{\theta,\xi\mapsto X}(\rho_{i\nu})(x_{i\nu}))_{\nu<n_i} \to X(C_i\vec{x}_i)))_{i<k},
\end{aligned}$$

where $\hat{\mu}$ is an intermediate symbol which is instantiated by μ or ν. An inductive predicate defined by tl from τ is called the *totality predicate* of τ, written T_τ, and a coinductive predicate is called the *cototality predicate* of τ, written ${}^{\mathrm{co}}T_\tau$. When a (co)totality predicate of type τ has parameter predicate variables $\vec{P}$ associated with free variables in τ, we can explicitly call it a *relativised (co)totality predicate*, written RT_τ (${}^{\mathrm{co}}RT_\tau$).

If there are n occurrences of μ in the definition of a type τ, there are 2^n variations of combinations of inductive definitions and coinductive definitions due to the way of instantiating $\hat{\mu}$ in $\mathrm{tl}_\theta(\tau)$. For a nested algebra ι depending on another algebra ι', we can say the ι-(co)totality ι'-(co)totality predicate, if there is no confusion. For example, the cototality predicate of $\mathbf{Nt}$ which internally uses the totality predicate of $\mathbf{L}_\alpha$ can be called the $\mathbf{Nt}$-cototality $\mathbf{L}_{\mathbf{Nt}}$-totality predicate.

For an algebra $\iota_{\vec{\alpha}}$, we define the structure-totality and the structure-cototality predicates $ST_{\iota_{\vec{\alpha}}}$ and ${}^{\mathrm{co}}ST_{\iota_{\vec{\alpha}}}$.

Definition 2.3.20 (Structure-totality and -cototality predicates)**.** Assume τ is a type and θ is a mapping from each free variable α in τ into a comprehension term $\{x^\alpha \mid \mathbf{T}\}$. We define the structure-totality and cototality predicates, ST_τ and ${}^{\mathrm{co}}ST_\tau$, of τ by instantiating $\hat{\mu}$ in $\mathrm{tl}_\theta(\tau)$ by μ and ν, respectively, and identifying formulas $\mathbf{T} \to A$ and A.

Example 2.3.21 (Totality and cototality predicates on $\mathbf{N}$ and $\mathbf{Nt}$)**.** We obtain from $\mathrm{tl}(\mathbf{N})$ the totality and cototality predicates $T_{\mathbf{N}} = \mu_X(X\,\mathsf{0}, \forall_n(X\,n \to X(\mathsf{S}\,n)))$ and ${}^{\mathrm{co}}T_{\mathbf{N}} = \nu_X(X\,\mathsf{0}, \forall_n(X\,n \to X(\mathsf{S}\,n)))$. We obtain from $\mathrm{tl}(\mathbf{Nt})$ the four variations of totality and cototality predicates, the $\mathbf{Nt}$-total $\mathbf{L}_{\mathbf{Nt}}$-total predicate, the $\mathbf{Nt}$-cototal $\mathbf{L}_{\mathbf{Nt}}$-total predicate, the $\mathbf{Nt}$-total $\mathbf{L}_{\mathbf{Nt}}$-cototal predicate and the $\mathbf{Nt}$-cototal $\mathbf{L}_{\mathbf{Nt}}$-cototal predicate.

Let RT_X be the relativised totality predicate on $\mathbf{L}_\alpha$. The introduction and elimination axioms of the $\mathbf{Nt}$-total $\mathbf{L}_{\mathbf{Nt}}$-total predicate $T_{\mathbf{Nt}}$ are

$$\forall_{as}(RT_{T_{\mathbf{Nt}}}\,as \to T_{\mathbf{Nt}}(\mathsf{Br}\,as)), \qquad T^+_{\mathbf{Nt}}$$
$$\forall_a(T_{\mathbf{Nt}}\,a \to \forall_{as}(RT_{T_{\mathbf{Nt}}\cap Q}\,as \to Q(\mathsf{Br}\,as)) \to Q\,a). \qquad T^-_{\mathbf{Nt}}$$

The clause and greatest-fixed-point axioms of the $\mathbf{Nt}$-cototal $\mathbf{L}_{\mathbf{Nt}}$-total predicate ${}^{\mathrm{co}}T_{\mathbf{Nt}}$ are

$$\forall_a({}^{\mathrm{co}}T\, a \to \exists_{as}(T_{{}^{\mathrm{co}}T}\, as \land a \operatorname{eqd} \mathsf{Br}\, as)) \qquad {}^{\mathrm{co}}T^-_{\mathbf{Nt}}$$
$$\forall_a(Q\, a \to \forall_a(Q\, a \to \exists_{as}(T_{({}^{\mathrm{co}}T \cup Q)}\, as \land a \operatorname{eqd} \mathsf{Br}\, as)) \to {}^{\mathrm{co}}T\, a). \qquad {}^{\mathrm{co}}T^+_{\mathbf{Nt}}$$

Example 2.3.22 (Structure-totality and cototality predicates of $\mathbf{L}_\alpha$)**.** Let α be a type variable and θ be a mapping $(\alpha \mapsto \{x^\alpha \mid \mathbf{T}\})$. From $\mathrm{tl}_\theta(\mathbf{L}_\alpha)$ we obtain $\hat{\mu}_X(X\,[], \forall_{x,xs}(\mathbf{T} \to X\, xs \to X(x::xs)))$ which yields $ST_{\mathbf{L}} = \mu_X(X\,[], \forall_{x,xs}(X\, xs \to X(x::xs)))$ and ${}^{\mathrm{co}}ST_{\mathbf{L}} = \nu_X(X\,[], \forall_{x,xs}(X\, xs \to X(x::xs)))$.

2.3.4 Monotonicity

Suppose that the occurrences of predicate variables $\vec{X}$ in a formula $A(\vec{X})$ are strictly positive, i.e., $\mathrm{F}(A(\vec{X}))$ and $\mathrm{SP}_{\vec{X}}(A(\vec{X}))$, then $A(\vec{X})$ satisfies *the monotonicity property*; $A(\vec{P})$ implies $A(\vec{Q})$ provided $\forall_{\vec{x}}(P_i\vec{x} \to Q_i\vec{x})$ for each i. This lemma plays an important role in proof normalization involving nestedness. In Section 2.5.3, we prove that the map operator is a realizer of the monotonicity lemma.

Lemma 2.3.23 (Monotonicity)**.** *Let $A(\vec{X})$ be a formula, n be the length of $\vec{X}$, and $\vec{P}$, $\vec{Q}$ be a list of predicates of length n. Assume $SP_{\vec{X}}(A(\vec{X}))$, then $A(\vec{X})$ satisfies the monotonicity property, namely, $\forall_{\vec{x}}(A(\vec{P}) \to (\forall_{\vec{x}}(P_i\vec{x} \to Q_i\vec{x}))_{i<n} \to A(\vec{Q}))$.*

Proof. By simultaneous induction on the construction of A.

Case $A(\vec{X}) = \forall_{\tilde{x}}\tilde{A}(\vec{X})$. We prove $\forall_{\vec{x}}(\forall_{\tilde{x}}\tilde{A}(\vec{P}) \to (\forall_{\vec{x}}(P_i\vec{x} \to Q_i\vec{x}))_{i<n} \to \forall_{\tilde{x}}\tilde{A}(\vec{Q}))$. Let $\vec{x}$ be given and assume $\forall_{\tilde{x}}\tilde{A}(\vec{P})$, $\forall_{\vec{x}}(P_i\vec{x} \to Q_i\vec{x})$ for each $i < n$, and also let $\tilde{x}$ be given. By induction hypothesis, it suffices to prove $\tilde{A}(\vec{P})$, which comes from the assumption $\forall_{\tilde{x}}\tilde{A}(\vec{P})$.

Case $A(\vec{X}) = B \to \tilde{A}(\vec{X})$. $\mathrm{FPV}(B) \cap \vec{X} = \emptyset$. Assume $B \to \tilde{A}(\vec{P})$, $\forall_{\vec{x}}(P_i\vec{x} \to Q_i\vec{x})$ for each $i < n$ and B, then we prove $\tilde{A}(\vec{Q})$. Using the induction hypothesis, it suffices to prove $\tilde{A}(\vec{P})$, which comes from the assumption $B \to \tilde{A}(\vec{P})$ and B.

Case $A(\vec{X}) = \tilde{P}\vec{t}$. We proceed by side induction on the construction of $\mathrm{Pred}(\tilde{P})$.

Side case $\tilde{P} = Z$. If $Z = X_j$ for some j, then the goal is

$$\forall_{\vec{x}}(P_j\vec{t} \to (\forall_{\vec{x}}(P_i\vec{x} \to Q_i\vec{x}))_{i<n} \to Q_j\vec{t}).$$

Use $\forall_{\vec{x}}(P_j\vec{x} \to Q_j\vec{x})$ in the premise. Otherwise, the goal is of the form $\forall_{\vec{x}}(Z\vec{t} \to (\forall_{\vec{x}}(P_i\vec{x} \to Q_i\vec{x}))_{i<n} \to Z\vec{t})$, which is trivial.

Side case $\tilde{P} = \{\vec{x} \mid \tilde{A}(\vec{x})\}$. The induction hypothesis is exactly the conclusion we desire.

Side case $\tilde{P} = I(\vec{R}(\vec{X}))$ where I is an inductive predicate $\mu_Z(\vec{K})$ and $\vec{R}$ is a list of predicates. Let n' be the length of $\vec{R}$. We prove

$$\forall_{\vec{x}}(I(\vec{R}(\vec{P}))\vec{t} \to (\forall_{\vec{x}}(P_i\vec{x} \to Q_i\vec{x}))_{i<n} \to I(\vec{R}(\vec{Q}))\vec{t}).$$

If $n' = 0$, it is trivial. Otherwise, let $\vec{x}$ be given and assume $I(\vec{R}(\vec{P}))\vec{t}$, say $I_{\vec{R}(\vec{P})}\vec{t}$ in short, and $\forall_{\vec{x}}(P_i\vec{x} \to Q_i\vec{x})$ for each $i < n$. We prove the goal $I_{\vec{R}(\vec{Q})}\vec{t}$ by using I^- for $I_{\vec{R}(\vec{P})}\vec{t}$.

Suppose that the clause formulas of $I_{\vec{X}}$ are $K_i = \forall_{\vec{x}}((A_{i\nu}(\vec{X}, Z))_{\nu<n_i} \to Z\vec{t})$, then the step formulas are $\forall_{\vec{x}}((A_{i\nu}(\vec{R}(\vec{P}), I_{\vec{R}(\vec{P})} \cap I_{\vec{R}(\vec{Q})}))_{\nu<n_i} \to I_{\vec{R}(\vec{Q})}\vec{t})$ for each $i < k$. Let $\vec{x}$ be given and assume $(A_{i\nu}(\vec{R}(\vec{P}), I_{\vec{R}(\vec{P})} \cap I_{\vec{R}(\vec{Q})}))_{\nu<n_i}$. Using I_i^+ to prove $I_{\vec{R}(\vec{Q})}\vec{t}$, it remains to prove $A_{i\nu}(\vec{R}(\vec{Q}), I_{\vec{R}(\vec{Q})})$ for $\nu < n_i$. We use induction hypothesis for $\mathrm{F}(A_{i\nu}(\vec{X}, Z))$,

$$\forall_{\vec{x}}(A_{i\nu}(\vec{R}(\vec{P}), I_{\vec{R}(\vec{P})} \cap I_{\vec{R}(\vec{Q})}) \to (\forall_{\vec{x}}(R_i(\vec{P})\vec{x} \to R_i(\vec{Q})\vec{x}))_{i<n'} \to$$
$$\forall_{\vec{x}}((I_{\vec{R}(\vec{P})} \cap I_{\vec{R}(\vec{Q})})\vec{x} \to I_{\vec{R}(\vec{Q})}\vec{x}) \to A_{i\nu}(\vec{R}(\vec{Q}), I_{\vec{R}(\vec{Q})})).$$

We already have the first premise. The second ones are by side induction hypotheses on $\mathrm{Pred}(R_i(\vec{X}))$. The third one is trivial from the definition of conjunction.

Side case $\tilde{P} = {}^{\mathrm{co}}I(\vec{R}(\vec{X}))$ where ${}^{\mathrm{co}}I(\vec{Y})$ is a coinductive predicate $\nu_Z(\vec{K})$ and $\vec{R}(\vec{X})$ is a list of predicates. We prove

$$\forall_{\vec{x}}({}^{\mathrm{co}}I(\vec{R}(\vec{P}))\vec{t} \to (\forall_{\vec{x}}(P_j\vec{x} \to Q_j\vec{x}))_{j<n} \to {}^{\mathrm{co}}I(\vec{R}(\vec{Q}))\vec{t}).$$

Suppose that the length of $\vec{R}$ is non-zero. Let $\vec{x}$ be given and assume ${}^{\mathrm{co}}I_{\vec{R}(\vec{P})}\vec{t}$, and $\forall_{\vec{x}}(P_i\vec{x} \to Q_i\vec{x})$ for each $i < n$. We prove the goal ${}^{\mathrm{co}}I_{\vec{R}(\vec{Q})}\vec{t}$ by using ${}^{\mathrm{co}}I^+$ with the competitor predicate ${}^{\mathrm{co}}I_{\vec{R}(\vec{P})}$. Suppose the same $\vec{K}$ as in the same way as the previous side case, then the costep formula is $\forall_{\vec{x}}({}^{\mathrm{co}}I_{\vec{R}(\vec{P})}\vec{x} \to \bigvee_{i<k} \exists_{\vec{y}}(\bigwedge \vec{E}_i \wedge \bigwedge_{\nu<n_i} A_{i\nu}(\vec{R}(\vec{Q}), {}^{\mathrm{co}}I_{\vec{R}(\vec{Q})} \cup {}^{\mathrm{co}}I_{\vec{R}(\vec{P})})))$. Let $\vec{x}$ be given and assume ${}^{\mathrm{co}}I_{\vec{R}(\vec{P})}\vec{x}$. Applying ${}^{\mathrm{co}}I^-$ to ${}^{\mathrm{co}}I_{\vec{R}(\vec{P})}\vec{x}$, then $\bigvee_{i<k} \exists_{\vec{y}}(\bigwedge \vec{E}_i \wedge \bigwedge_{\nu<n_i} A_{i\nu}(\vec{R}(\vec{P}), {}^{\mathrm{co}}I_{\vec{R}(\vec{P})}))$ holds, and use $\vee^-$ for this disjunction. For the case of i-th disjunct, assume $\exists_{\vec{y}}(\bigwedge \vec{E}_i \wedge \bigwedge_{\nu<n_i} A_{i\nu}(\vec{R}(\vec{P}), {}^{\mathrm{co}}I_{\vec{R}(\vec{P})})$. By means of $\vee_0^+$, $\vee_1^+$, $\exists^-$, $\exists^+$, $\wedge^-$ and $\wedge^+$, we end up with proving $A_{i\nu}(\vec{R}(\vec{Q}), {}^{\mathrm{co}}I_{\vec{R}(\vec{Q})} \cup {}^{\mathrm{co}}I_{\vec{R}(\vec{P})})$ from $A_{i\nu}(\vec{R}(\vec{P}), {}^{\mathrm{co}}I_{\vec{R}(\vec{P})})$. By the induction hypothesis for $\mathrm{F}(A_{i\nu}(\vec{X}, Z))$,

$$\forall_{\vec{x}}(A_{i\nu}(\vec{R}(\vec{P}), {}^{\mathrm{co}}I_{\vec{R}(\vec{P})}) \to (\forall_{\vec{x}}(R_i(\vec{P})\vec{x} \to R_i(\vec{Q})\vec{x}))_{i<m} \to$$
$$\forall_{\vec{x}}({}^{\mathrm{co}}I_{\vec{R}(\vec{P})}\vec{x} \to ({}^{\mathrm{co}}I_{\vec{R}(\vec{Q})} \cup {}^{\mathrm{co}}I_{\vec{R}(\vec{P})})\vec{x}) \to A_{i\nu}(\vec{R}(\vec{Q}), {}^{\mathrm{co}}I_{\vec{R}(\vec{Q})} \cup {}^{\mathrm{co}}I_{\vec{R}(\vec{P})})).$$

The first premise is in the assumption and the second and the third ones are from the side induction hypothesis as in the previous side case. □

2.3.5 Simultaneous Definitions

We consider a generalization of predicate definitions to adopt *simultaneously defined predicates*. We modify Definition 2.3.5, in the following way.

Definition 2.3.24 (Formulas and predicates with simultaneity)**.** Let $\vec{X}$ be a list of predicate variables of length l and j be a monotone surjective function from $\{0, \ldots, k-1\}$ to $\{0, \ldots, l-1\}$. We define predicate forms P by

$$P ::= X \mid \{\vec{x} \mid A\} \mid \mu_{\vec{X}}(\forall_{\vec{x}_i}((A_{i\nu})_{\nu<n_i} \to X_{j_i}\vec{t}_i))_{i<k} \mid \nu_{\vec{X}}(\forall_{\vec{x}_i}((A_{i\nu})_{\nu<n_i} \to X_{j_i}\vec{t}_i))_{i<k}.$$

Let J be $\mu_{\vec{X}}(\forall_{\vec{x}_i}((A_{i\nu})_{\nu<n_i} \to X_{j_i}\vec{t}_i))_{i<k})$ or $\nu_{\vec{X}}(\forall_{\vec{x}_i}((A_{i\nu})_{\nu<n_i} \to X_{j_i}\vec{t}_i))_{i<k})$. The definition of free predicate variables FPV is defined also for the above predicate forms.

$$\mathrm{FPV}(J) := \bigcup_{i<k}\bigcup_{\nu<n_i}\mathrm{FPV}(A_{i\nu}) \setminus \{\vec{X}\}.$$

The strictly positive occurrences of predicate variables are modified in the following way.

$$\frac{\text{for all } i<k, \text{ for all } \nu<n_i,\ \mathrm{SP}_{\vec{Y},\vec{X}}(A_{i\nu})}{\mathrm{SP}_{\vec{Y}}(J)}\ .$$

The inhabited predicate forms are modified as follows. Here we mean the m-th tail of $\vec{X}$ by $\vec{X}^{(m)}$.

$$\frac{\text{for all } m<l, \text{ there exists } i,\ j_i=m \text{ and for all } \nu<n_i,\ \mathrm{Inhab}_{\vec{Y},\vec{X}^{(m)}}(A_{i\nu})}{\mathrm{Inhab}_{\vec{Y}}(J)}\ .$$

The predicates are defined as follows.

$$\frac{\text{for all } i<k,\ \nu<n_i,\ \mathrm{F}(A_{i\nu}) \text{ and } \mathrm{SP}_X(A_{i\nu}) \quad \mathrm{Inhab}(J)}{\mathrm{Pred}(J)}\ .$$

The introduction and elimination axioms are given in a similar way as the non-simultaneous case.

Definition 2.3.25 (Introduction and elimination axioms of simultaneous inductive definitions)**.** Let $\vec{X}$ be a list of predicate variables of length l and $\vec{I}$ be a list of simultaneously inductively defined predicates $\mu_{\vec{X}}(\forall_{\vec{x}_i}((A_{i\nu}(\vec{X}))_{\nu<n_i} \to X_{j_i}\vec{t}_i))_{i<k})$. The introduction axioms of $\vec{I}$ are given as follows.

$$\forall_{\vec{x}_i}((A_{i\nu}(\vec{I}))_{\nu<n_i} \to I_{j_i}\vec{t}_i)). \tag{$\vec{I})^+_i$}$$

Let $\vec{Q}$ be a list of predicate variables of length l such that the arity of Q_m is same as the one of I_m for each $m<l$, and $\vec{P}$ be a list of predicates of the form $I_m \cap Q_m$. For each $m<l$ we define the elimination axioms I^-_m as follows.

$$\forall_{\vec{x}}(I_m\,\vec{x} \to (\forall_{\vec{x}_i}((A_{i\nu}(\vec{P}))_{\nu<n_i} \to Q_{j_i}\vec{t}_i))_{i<k} \to Q_m\,\vec{x}). \tag{$\vec{I})^-_m$}$$

Definition 2.3.26 (Closure and greatest-fixed-point axioms of simultaneous coinductive definitions)**.** Let $\vec{X}$ be a list of predicate variables of length l and $\vec{{}^{\mathrm{co}}I}$ be a list of simultaneously coinductively defined predicates $\nu_{\vec{X}}(\forall_{\vec{x}_i}((A_{i\nu}(\vec{X}))_{\nu<n_i} \to X_{j_i}\vec{t}_i))_{i<k})$. Assume variables $\vec{y}_i$ for each of $\vec{t}_i$ to form an equality $E_{ii'} := y_{ii'}$ eqd $t_{ii'}$. For each $m<l$ we find i_m and i'_m such that $i_m \le i < i'_m$ implies $j_i = m$. For each $m<l$ we define the closure axioms $(\vec{{}^{\mathrm{co}}I})^-_m$ as follows.

$$\forall_{\vec{y}}({}^{\mathrm{co}}I_m\vec{y} \to \bigvee_{i;\, i_m\le i<i'_m}\exists_{\vec{x}_i}(\textstyle\bigwedge_{\nu<n_i} A_{i\nu}(\vec{{}^{\mathrm{co}}I}) \land \bigwedge \vec{E}_i)). \tag{$\vec{{}^{\mathrm{co}}I})^-_m$}$$

Let $\vec{Q}$ be a list of predicate variables of length l such that the arity of Q_m is same as the one of I_m for each $m < l$, and $\vec{P}$ be a list of predicates of the form ${}^{\mathrm{co}}I_m \cup Q_m$. For each $m < l$ we define the greatest-fixed-point axioms $({}^{\mathrm{co}}\vec{I})^+_m$ as follows.

$$\forall_{\vec{x}}(Q_m\,\vec{x} \to (\forall_{\vec{y}}({}^{\mathrm{co}}I_m\vec{y} \to \bigvee_{i;\, i_m \le i < i'_m} \exists_{\vec{x}_i}(\bigwedge_{\nu<n_i} A_{i\nu}(\vec{P}) \land \bigwedge \vec{E}_i)))_{m<l} \to {}^{\mathrm{co}}I_m\,\vec{x}). \qquad ({}^{\mathrm{co}}\vec{I})^+_m$$

Consider the algebras $(\mathbf{Ts}, \mathbf{T})$ of tree lists and trees defined in Example 2.1.18. We define the totality predicates and the cototality predicates of them for example.

Example 2.3.27. Consider the simultaneously defined algebras of tree lists and trees $(\mathbf{Ts}, \mathbf{T})$. We simultaneously define the predicates of $(\mathbf{Ts}, \mathbf{T})$-totality. The $(\mathbf{Ts}, \mathbf{T})$-totality predicates are $(T_{\mathbf{Ts}}, T_{\mathbf{T}})$ whose arities are $(\mathbf{Ts})$ and $(\mathbf{T})$, respectively. Define $(T_{\mathbf{Ts}}, T_{\mathbf{T}})$ to be $\mu_{X,Y}(K_0, K_1, K_2)$ where

$$\begin{aligned}
K_0 &= X\,\mathsf{Empty},\\
K_1 &= \forall_a(Y\,a \to \forall_{as}(X\,as \to X(\mathsf{Tcons}\,a\,as))),\\
K_2 &= \forall_{as}(Y\,as \to X(\mathsf{Branch}\,as)).
\end{aligned}$$

The introduction and elimination axioms are as follows. Let P_0 and P_1 be predicate variables of arities $(\mathbf{Ts})$ and $(\mathbf{T})$.

$$\begin{aligned}
&T_{\mathbf{Ts}}\,\mathsf{Empty}, && (T_{\mathbf{Ts}}, T_{\mathbf{T}})^+_0\\
&\forall_a(T_{\mathbf{T}}\,a \to \forall_{as}(T_{\mathbf{Ts}}\,as \to T_{\mathbf{Ts}}(\mathsf{Tcons}\,a\,as))), && (T_{\mathbf{Ts}}, T_{\mathbf{T}})^+_1\\
&\forall_{as}(T_{\mathbf{Ts}}\,as \to T_{\mathbf{T}}(\mathsf{Branch}\,as)), && (T_{\mathbf{Ts}}, T_{\mathbf{T}})^+_2\\
&\forall_{as}(T_{\mathbf{Ts}}\,as \to S_0 \to S_1 \to S_2 \to P_0\,as), && (T_{\mathbf{Ts}}, T_{\mathbf{T}})^-_0\\
&\forall_a(T_{\mathbf{T}}\,a \to S_0 \to S_1 \to S_2 \to P_1\,a), && (T_{\mathbf{Ts}}, T_{\mathbf{T}})^-_1
\end{aligned}$$

where

$$\begin{aligned}
S_0 &= P_0\,\mathsf{Empty},\\
S_1 &= \forall_a(T_{\mathbf{T}}\,a \to P_1\,a \to \forall_{as}(T_{\mathbf{Ts}}\,as \to P_0\,as \to P_0(\mathsf{Tcons}\,a\,as))),\\
S_2 &= \forall_{as}(T_{\mathbf{Ts}}\,as \to P_0\,as \to P_1\,as)).
\end{aligned}$$

The companion predicates of the above are simultaneously coinductively defined predicates $({}^{\mathrm{co}}T_{\mathbf{Ts}}, {}^{\mathrm{co}}T_{\mathbf{T}})$ of arities $(\mathbf{Ts})$ and $(\mathbf{T})$, respectively. They are defined to be $\nu_{X,Y}(K_0, K_1, K_2)$, then closure and greatest-fixed-point axioms are as follows.

$$\begin{aligned}
&\forall_{as}({}^{\mathrm{co}}T_{\mathbf{Ts}}\,as \to as\,\mathrm{eqd}\,\mathsf{Empty} \lor\\
&\qquad \exists_{b,bs}({}^{\mathrm{co}}T_{\mathbf{T}}\,b \land {}^{\mathrm{co}}T_{\mathbf{Ts}}\,bs \land as\,\mathrm{eqd}\,\mathsf{Tcons}\,b\,bs)), && ({}^{\mathrm{co}}T_{\mathbf{Ts}}, {}^{\mathrm{co}}T_{\mathbf{T}})^-_0\\
&\forall_a({}^{\mathrm{co}}T_{\mathbf{T}}\,a \to \exists_{as}({}^{\mathrm{co}}T_{\mathbf{Ts}}\,as \land a\,\mathrm{eqd}\,\mathsf{Branch}\,as)), && ({}^{\mathrm{co}}T_{\mathbf{Ts}}, {}^{\mathrm{co}}T_{\mathbf{T}})^-_1\\
&\forall_{as}(P_0\,as \to {}^{\mathrm{co}}S_0 \to {}^{\mathrm{co}}S_1 \to {}^{\mathrm{co}}T_{\mathbf{Ts}}\,as), && ({}^{\mathrm{co}}T_{\mathbf{Ts}}, {}^{\mathrm{co}}T_{\mathbf{T}})^+_0\\
&\forall_a(P_1\,as \to {}^{\mathrm{co}}S_0 \to {}^{\mathrm{co}}S_1 \to {}^{\mathrm{co}}T_{\mathbf{T}}\,a), && ({}^{\mathrm{co}}T_{\mathbf{Ts}}, {}^{\mathrm{co}}T_{\mathbf{T}})^+_1
\end{aligned}$$

where,

$${}^{\mathrm{co}}S_0 = \forall_{as}(P_0\,as \to as\,\mathrm{eqd}\,\mathsf{Empty} \lor \exists_{b,bs}(({}^{\mathrm{co}}T_{\mathbf{T}} \cup P_1)b \land ({}^{\mathrm{co}}T_{\mathbf{Ts}} \cup P_0)bs \land as\,\mathrm{eqd}\,\mathsf{Tcons}\,b\,bs)),$$

$${}^{\mathrm{co}}S_1 = \forall_a(P_1\,a \to \exists_{as}(({}^{\mathrm{co}}T_{\mathbf{Ts}} \cup P_0)as \land a\,\mathrm{eqd}\,\mathsf{Branch}\,as)).$$

2.4 Proofs

In this section we focus on providing our notion of *proofs* or *derivations* in TCF. The word derivation is used for a formal representation of a proof, while the word proof is rather used for the informal level. We formulate the term representation of natural deduction derivations and its conversion. We leave the details of first order logic to the relevant literature [vD94, TS00, SW12].

2.4.1 Natural Deduction

Our base logic is first-order minimal logic with the implication and the universal quantifier. Axioms given by inductive and coinductive definitions extend the logic. We focus on defining the term representation of derivations which is required in program extraction in Section 2.5. We define the notion of derivations.

Definition 2.4.1 (Derivations)**.** Let A and B be variables which range over formulas, x range over object variables and r range over terms. We inductively define a derivation.

Assumption An arbitrary formula A is a derivation from an open assumption A.

Axiom An arbitrary axiom A is a derivation.

We depict this derivation by A. There are the introduction and elimination rules for the implication and for the universal quantifier.

The introduction rule of the universal quantifier We assume a derivation of A which does not contain any open assumption which has x as a free variable and construct a derivation of $\forall_x A$.

The elimination rule of the universal quantifier We assume a derivation of $\forall_x A$ and a term r of the same type as x and construct a derivation of A.

The introduction rule of the implication We assume a derivation of B and construct a derivation of $A \to B$ with cancelling all open assumption A in the derivation of B.

The elimination rule of the implication We assume a derivation of $A \to B$ and another one of B and construct a derivation of B.

Those derivations are depicted as follows. The introduction rules of the universal quantifier and the implication are denoted by $\forall^+$ and $\to^+$ and the elimination rules of the universal quantifier and the implication are denoted by $\forall^-$ and $\to^-$, respectively. A use of the introduction of the implication and the corresponding cancelled open assumptions are indicated by an assumption variable and square brackets.

Via the Curry-Howard correspondence we consider *derivation terms*, a term representation of derivations. We consider each axiom name as a constant of derivation terms. For example, let I and ${}^{\mathrm{co}}I$ be inductively and coinductively defined predicate constants, respectively, then I_i^+, I^-, ${}^{\mathrm{co}}I^-$ and ${}^{\mathrm{co}}I^+$, are constants. Let c range over constants.

Definition 2.4.2 (Derivation terms)**.** We define derivation terms by induction on the construction of a derivation.

(*Assumption*) A derivation term of a derivation $u : A$ is u^A.

$$\frac{\begin{array}{c}\vdots\, D\\ A\end{array}}{\forall_x A}\ \forall^+ \qquad\qquad \frac{\begin{array}{c}\vdots\, D\\ \forall_{x^\rho} A\end{array}\quad r^\rho}{A(r)}\ \forall^-$$

$$\frac{\begin{array}{c}[u:A]\\ \vdots\, D\\ B\end{array}}{A \to B}\ \to^+ u \qquad\qquad \frac{\begin{array}{cc}\vdots\, D & \vdots\, E\\ A \to B & A\end{array}}{B}\ \to^-$$

($Axiom$) A derivation term of a derivation A by an axiom c is c^A.
($\forall^+$) A derivation term of a derivation

$$\frac{\begin{array}{c}\vdots\, D\\ A\end{array}}{\forall_x A}\ \forall^+$$

is $(\lambda_x M)^{\forall_x A}$, where M^A is a derivation term of D.
($\forall^-$) A derivation term of a derivation

$$\frac{\begin{array}{c}\vdots\, D\\ \forall_{x^\rho} A\end{array}\quad r^\rho}{A(r)}\ \forall^-$$

is $(M^{\forall_x A} r)^A$, where $M^{\forall_x A}$ is a derivation term of D.
($\to^+$) A derivation term of a derivation

$$\frac{\begin{array}{c}[u:A]\\ \vdots\, D\\ B\end{array}}{A \to B}\ \to^+ u$$

is $(\lambda_x^A M^B)^{A\to B}$, where M^B is a derivation term of D.
($\to^-$) A derivation term of a derivation

$$\frac{\begin{array}{cc}\vdots\, D & \vdots\, E\\ A \to B & A\end{array}}{B}\ \to^-$$

is $(M^{A\to B} N^A)^B$, where $M^{A\to B}$ and N^A are the derivation terms of D and E, respectively.

Example 2.4.3. Recall Even and $T_{\mathbf{N}}$ defined in Example 2.3.11 and Example 2.3.21, respectively. We prove $\forall_n(\mathrm{Even}\, n \to T_{\mathbf{N}}\, n)$ as follows. Suppose that S_0 and S_1 are the step

formulas of Even^-, i.e., $P\,\mathsf{0}$ and $\forall_n(\mathrm{Even}\,n \to P\,n \to P\,(\mathsf{S}(\mathsf{S}\,n)))$. Then,

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\forall_n(\mathrm{Even}\,n \to S_0 \to S_1 \to T_{\mathbf{N}}\,n)}\,\mathrm{Even}^- \quad n}{\mathrm{Even}\,n \to S_0 \to S_1 \to T_{\mathbf{N}}\,n} \quad [\mathrm{Even}\,n]^u}{S_0 \to S_1 \to T_{\mathbf{N}}\,n} \quad \begin{matrix}\vdots\\ S_0\end{matrix} \quad \begin{matrix}\vdots\\ M_0\end{matrix}}{S_1 \to T_{\mathbf{N}}\,n} \quad \begin{matrix}\vdots\\ S_1\end{matrix} \quad \begin{matrix}\vdots\\ M_1\end{matrix}}{T_{\mathbf{N}}\,n}}{\mathrm{Even}\,n \to T_{\mathbf{N}}\,n}\,{\to^+}\,u}{\forall_n(\mathrm{Even}\,n \to T_{\mathbf{N}}\,n)},$$

where M_0 and M_1 are

$$\dfrac{}{T_{\mathbf{N}}\,\mathsf{0}}\,(T_{\mathbf{N}})_0^+,$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\forall_n(T_{\mathbf{N}}\,n \to T_{\mathbf{N}}(\mathsf{S}\,n))}\,(T_{\mathbf{N}})_1^+ \quad \mathsf{S}\,n}{T_{\mathbf{N}}(\mathsf{S}\,n) \to T_{\mathbf{N}}(\mathsf{S}(\mathsf{S}\,n))} \quad \dfrac{\dfrac{}{\forall_n(T_{\mathbf{N}}\,n \to T_{\mathbf{N}}(\mathsf{S}\,n))}\,(T_{\mathbf{N}})_1^+ \quad n}{T_{\mathbf{N}}\,n \to T_{\mathbf{N}}(\mathsf{S}\,n)} \quad [T_{\mathbf{N}}\,n]^{v_1}}{T_{\mathbf{N}}(\mathsf{S}\,n)}}{T_{\mathbf{N}}(\mathsf{S}(\mathsf{S}\,n))}}{T_{\mathbf{N}}\,n \to T_{\mathbf{N}}(\mathsf{S}(\mathsf{S}\,n))}\,{\to^+}\,v_1}{\mathrm{Even}\,n \to T_{\mathbf{N}}\,n \to T_{\mathbf{N}}(\mathsf{S}(\mathsf{S}\,n))}\,{\to^+}\,v_0}{\forall_n(\mathrm{Even}\,n \to T_{\mathbf{N}}\,n \to T_{\mathbf{N}}(\mathsf{S}(\mathsf{S}\,n)))}.$$

Omitting the superscript formulas, the derivation term of the above is

$$\lambda_{n,u}(\mathrm{Even}^-\,n\,u\,(T_{\mathbf{N}})_0^+\,(\lambda_{n,v_0,v_1}(T_{\mathbf{N}})_1^+(\mathsf{S}\,n)((T_{\mathbf{N}})_1^+\,n\,v_1))).$$

2.4.2 Proof Conversion

We adopt the standard conversion rules for $\forall$ and $\to$. In addition, we extend the conversion in order to support axioms of inductively and coinductively defined predicates.

Definition 2.4.4 (Proof conversion for $\to$ and $\forall$)**.** The conversions for $\to$ and $\forall$ are defined to be

$$\dfrac{\dfrac{\begin{matrix}[u:A]\\ \vdots\\ B\end{matrix}}{A \to B}\,{\to^+}\,u \quad \begin{matrix}\vdots\\ A\end{matrix} \quad \begin{matrix}\vdots\\ N\end{matrix}}{B}\,{\to^-} \quad \mapsto \quad \begin{matrix}\vdots\\ A\end{matrix} \quad \begin{matrix}\vdots\\ N\end{matrix} \quad \begin{matrix}\vdots\\ M\end{matrix} \quad \begin{matrix}\vdots\\ B\end{matrix},$$

$$\dfrac{\dfrac{\begin{matrix}\vdots\\ A(x)\end{matrix} \quad \begin{matrix}\vdots\\ M_x\end{matrix}}{\forall_x A(x)}\,\forall^+ \quad r}{A(r)}\,\forall^- \quad \mapsto \quad \begin{matrix}\vdots\\ A(r)\end{matrix} \quad \begin{matrix}\vdots\\ M_r\end{matrix}.$$

The following conversion rule removes a certain use of the introduction and elimination axioms which is taken as a detour.

Definition 2.4.5 (Proof conversion for inductive predicates)**.** Let I be an inductively defined predicate and P be a predicate variable of the same arity as I. Abbreviate the step formulas of I^- as $\vec{S}$ and suppose that each I_i^+ is of the form $\forall_{\vec{x}}((A_{i\nu}(I))_{\nu<n} \to I\vec{t})$. Then, we consider the following derivation:

$$\dfrac{\dfrac{\dfrac{}{\forall_{\vec{x}}(I\vec{x} \to \vec{S} \to P\vec{x})}\,I^-\quad \vec{t}}{I\vec{t} \to \vec{S} \to P\vec{t}}\quad \dfrac{\dfrac{\dfrac{}{\forall_{\vec{x}}((A_{i\nu}(I))_{\nu<n} \to I\vec{t})}\,I_i^+\quad \vec{x}}{(A_{i\nu}(I))_{\nu<n} \to I\vec{t}}\quad \begin{matrix}\vdots\, M_{\nu<n}\\ (A_{i\nu}(I))_{\nu<n}\end{matrix}}{I\vec{t}}}{\vec{S} \to P\vec{t}}\quad \begin{matrix}\vdots\, \vec{N}\\ \vec{S}\end{matrix}}{P\vec{t}}\,.$$

Suppose that S_i is of the form $\forall_{\vec{x}}^{\mathrm{nc}}((A_{i\nu}(I\cap P))_{\nu<n} \to P\vec{t})$ and Mon abbreviates Lemma 2.3.23. We convert the above derivation into

$$\dfrac{\dfrac{\begin{matrix}\vdots\, N_i\\ \forall_{\vec{x}}((A_{i\nu}(I\cap P))_{\nu<n} \to P\vec{t})\end{matrix}\quad \vec{x}}{(A_{i\nu}(I\cap P))_{\nu<n} \to P\vec{t}}\quad \dfrac{\begin{matrix}\vdots\, K\\ \forall_{\vec{x}}(I\vec{x} \to (I\cap P)\vec{x}) \to A_{i\nu}(I\cap P)\end{matrix}\quad \begin{matrix}\vdots\, L\\ \forall_{\vec{x}}(I\vec{x} \to (I\cap P)\vec{x})\end{matrix}}{(A_{i\nu}(I\cap P))_{\nu<n}}}{P\vec{t}}\,,$$

where K and L are respectively as follows:

$$\dfrac{\dfrac{\dfrac{}{\forall_{\vec{x}}(A_{i\nu}(I) \to \forall_{\vec{x}}(I\vec{x} \to (I\cap P)\vec{x}) \to A_{i\nu}(I\cap P))}\,\mathrm{Mon}\quad \vec{x}}{A_{i\nu}(I) \to \forall_{\vec{x}}(I\vec{x} \to (I\cap P)\vec{x}) \to A_{i\nu}(I\cap P)}\quad \begin{matrix}\vdots\, M_\nu\\ A_{i\nu}(I)\end{matrix}}{\forall_{\vec{x}}(I\vec{x} \to (I\cap P)\vec{x}) \to A_{i\nu}(I\cap P)}\,,$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\forall_{\vec{x}}(I\vec{x} \to \vec{S} \to P\vec{x})}\,I^-\quad \vec{t}}{I\vec{t} \to \vec{S} \to P\vec{t}}\quad [u : I\vec{t}]}{\vec{S} \to P\vec{t}}\quad \begin{matrix}\vdots\, \vec{N}\\ \vec{S}\end{matrix}}{P\vec{t}}}{(I\cap P)\vec{t}}}{I\vec{t} \to (I\cap P)\vec{t}}\,{}^{[u : I\vec{t}]}{\to^+}\, u}{\forall_{\vec{x}}(I\vec{x} \to (I\cap P)\vec{x})}\,.$$

As a derivation term, the above conversion reads

$$I^-\,\vec{t}\,(I_i^+\,\vec{x}\,M_{\nu<n})\,\vec{N} \mapsto N_i\,\vec{t}(\mathrm{Mon}\,\vec{x}\,M_\nu(\lambda_{\vec{x},u}(\wedge^+u(I^-\,\vec{x}\;u\,\vec{N})))_{\nu<n}.$$

We consider conversion involving coinductively defined predicates. A certain use of a closure axiom and a greatest-fixed-point axiom is considered to be a detour and removed.

Definition 2.4.6 (Proof conversion for coinductive predicates)**.** Let ${}^{\mathrm{co}}I$ be a coinductively defined predicate and P be a predicate variable of the same arity of ${}^{\mathrm{co}}I$. We abbreviate $\bigvee_{i<k}(\bigwedge_{\nu<n} A_{i\nu}(X))$ as C_X and the costep formula of ${}^{\mathrm{co}}I^+$ as ${}^{\mathrm{co}}S$, then we consider the following derivation:

$$\dfrac{\dfrac{\dfrac{}{\forall_{\vec{x}}({}^{\mathrm{co}}I\vec{x} \to C_{{}^{\mathrm{co}}I})}\,{}^{\mathrm{co}}I^-\quad \vec{t}}{{}^{\mathrm{co}}I\vec{t} \to C_{{}^{\mathrm{co}}I}} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\forall_{\vec{x}}(P\vec{x} \to {}^{\mathrm{co}}S \to {}^{\mathrm{co}}I\vec{x})}\,{}^{\mathrm{co}}I^+\quad \vec{t}}{P\vec{t} \to {}^{\mathrm{co}}S \to {}^{\mathrm{co}}I\vec{t}} \quad \vdots\, N}{{}^{\mathrm{co}}S \to {}^{\mathrm{co}}I\vec{t}} \quad P\vec{t}}{{}^{\mathrm{co}}I\vec{t}} \quad \vdots\, M}{C_{{}^{\mathrm{co}}I}} \quad {}^{\mathrm{co}}S$$

.

By the conversion rule, the occurrence of ${}^{\mathrm{co}}I^-$ in the above derivation is removed.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\forall_{\vec{x}}(P\vec{x} \to \bigvee_{i<k}(\bigwedge_{\nu<n_i} A_{i\nu}({}^{\mathrm{co}}I \cup P)))}\quad \vec{t} \quad \vdots\, M}{P\vec{t} \to \bigvee_{i<k}(\bigwedge_{\nu<n_i} A_{i\nu}({}^{\mathrm{co}}I \cup P))} \quad \vdots\, N \quad P\vec{t}}{\bigvee_{i<k}(\bigwedge_{\nu<n_i} A_{i\nu}({}^{\mathrm{co}}I \cup P))} \qquad \dfrac{[u_i : \bigwedge_{\nu<n_i} A_{i\nu}({}^{\mathrm{co}}I \cup P)] \quad \vdots\, K_i \quad \bigwedge_{\nu<n_i} A_{i\nu}({}^{\mathrm{co}}I)}{\bigvee_{i<k}(\bigwedge_{\nu<n_i} A_{i\nu}({}^{\mathrm{co}}I))}\,\bigvee_i^+}{\bigvee_{i<k}(\bigwedge_{\nu<n_i} A_{i\nu}({}^{\mathrm{co}}I))}\,\bigvee_{\vec{u}}^-$$

,

where we suppose that A' is a formula $\forall_{\vec{x}}(({}^{\mathrm{co}}I \cup P)\vec{t} \to {}^{\mathrm{co}}I\vec{t})$ and B^i_X is a formula $\bigwedge_{\nu<n_i} A_{i\nu}(X)$. Omitting i without any confusion, each K_i is given as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\forall_{\vec{x}}(B_{{}^{\mathrm{co}}I \cup P} \to A' \to B_{{}^{\mathrm{co}}I})}\,\mathrm{Mon}\quad \vec{t}}{B_{{}^{\mathrm{co}}I \cup P} \to A' \to B_{{}^{\mathrm{co}}I}}}{A' \to B_{{}^{\mathrm{co}}I}} \qquad \dfrac{\dfrac{\dfrac{[w : {}^{\mathrm{co}}I\vec{t} \vee P\vec{t}] \quad [v_0 : {}^{\mathrm{co}}I\vec{t}] \quad {}^{\mathrm{co}}I\vec{t}}{{}^{\mathrm{co}}I\vec{t}}\,\vee^-_{v_0,v_1}}{({}^{\mathrm{co}}I \cup P)\vec{t} \to {}^{\mathrm{co}}I\vec{t}}\,{\to^+}\, w \quad [v_1 : P\vec{t}] \quad \vdots\, L_i}{A'}}{B_{{}^{\mathrm{co}}I}}$$

.

By using ${}^{\mathrm{co}}I^+$, we can derive ${}^{\mathrm{co}}I\vec{t}$ from $P\vec{t}$. Note that we can also use the derivation M for the costep formula of ${}^{\mathrm{co}}I^+$, then the derivation L_i is given as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\forall_{\vec{x}}(P\vec{x} \to {}^{\mathrm{co}}S \to {}^{\mathrm{co}}I\vec{x})}\,{}^{\mathrm{co}}I^+\quad \vec{t}}{P\vec{t} \to {}^{\mathrm{co}}S \to {}^{\mathrm{co}}I\vec{t}} \quad [v_1 : P\vec{t}]}{{}^{\mathrm{co}}S \to {}^{\mathrm{co}}I\vec{t}} \quad \vdots\, M}{{}^{\mathrm{co}}S}}{{}^{\mathrm{co}}I\vec{t}}$$

.

As a derivation term, the above conversion rule is

$${}^{\mathrm{co}}I^-\,\vec{t}\,({}^{\mathrm{co}}I^+\,\vec{t}\,N\,M) \mapsto \bigvee{}^-\,(MN)(\lambda_{u_i}(\mathrm{Mon}\,\vec{t}\,u_i\,\lambda_{\vec{x},w}(\vee^-\,w\,v_0({}^{\mathrm{co}}I^+\,\vec{x}\,v_1\,M))))_{i<n}.$$

2.5 Realizability Interpretation

According to Kreisel's modified realizability interpretation, a formula of the form $\forall_x A(x)$ is realized by a recursive function which maps a given x to a realizer of $A(x)$. Similarly, a formula of the form $A \to B$ is realized by a recursive function which maps a realizer of A to a realizer of B. In order to fine-tune computational content of proofs, we consider *computational* and *non-computational* universal quantifiers and implications, denoted by $\forall^{\mathrm{c}}$, $\forall^{\mathrm{nc}}$, $\to^{\mathrm{c}}$ and $\to^{\mathrm{nc}}$. These additional superscript notations are called *decorations*. The syntax of decorated universal quantifiers and implications are given by modifying Definition 2.3.1 and 2.3.5.

Definition 2.5.1 (Decorated formula forms)**.** We define formula forms by

$$A, B ::= P\vec{t} \mid A \to^{\mathrm{c}} B \mid \forall^{\mathrm{c}}_x A \mid A \to^{\mathrm{nc}} B \mid \forall^{\mathrm{nc}}_x A,$$

and $\forall$ and $\to$ in the definition of predicate forms are arbitrarily $\forall^{\mathrm{c}}$ or $\forall^{\mathrm{nc}}$, and $\to^{\mathrm{c}}$ or $\to^{\mathrm{nc}}$, respectively. Then the rest of definitions are modified to accept the same rules for decorated formula forms replacing the rules for non-decorated formula forms.

The non-computational universal quantifier was first studied by Berger [Ber93]. They are logically equivalent, but computationally different. The computational ones, namely, $\forall^{\mathrm{c}}$ and $\to^{\mathrm{c}}$, are interpreted by functions as commonly done. A realizer of $\forall^{\mathrm{nc}}_x A(x)$ is defined to be a realizer of $A(x)$ for an arbitrary x. The input x can occur in the formula $A(x)$, but not used to compute the output. A realizer of $A \to^{\mathrm{nc}} B$ is defined to be a realizer of B provided there exists a realizer of A. Here, an existence of a realizer of A is necessary, but it is not used.

Section 2.5.1 is devoted to discuss the decoration of clause formulas of inductive and coinductive definitions due to Ratiu and Schwichtenberg [RS10] and Schwichtenberg [SW12]. In Section 2.5.2, we introduce the notion of realizability interpretation for our setting based on Schwichtenberg [SW12]. We extend it to nested and coinductive definitions, then the program extraction is given. We prove the soundness of our realizability interpretation in Section 2.5.3. Theoretically, it claims that every theorem has a realizer.

2.5.1 Decorating Inductive and Coinductive Definitions

We consider computational inductive and coinductive definitions through decorating clause formulas explicitly. Then, we also study non-computational inductive and coinductive definitions. Although decorated formulas are logically the same as non-decorated formulas, they are computationally different. This difference becomes clear from the realizability interpretation given in Section 2.5.2. At the end of this section, we extend the notion of proofs to the inference rules of decorated connectives. Particularly in the introduction axioms of non-computational universal quantifier and implication we pose some restrictions in order to keep the computational consistency.

Definition 2.5.2 (Decorating Inductive Definitions)**.** Suppose that an inductive predicate I is given as

$$\mu_X(\forall_{\vec{x}}^{\mathrm{c/nc}}((A_{i\nu}(X))_{\nu<n_i} \to^{\mathrm{c/nc}} X\vec{t}))_{i<k},$$

where $\forall^{\mathrm{c/nc}}$ and $\to^{\mathrm{c/nc}}$ are arbitrarily decorated. The introduction axioms I_i^+ are obtained by substituting I for X in $A_{i\nu}$. The elimination axiom I^- is decorated as

$$\forall_{\vec{x}}^{\mathrm{nc}}(I\vec{x} \to^{\mathrm{c}} (\forall_{\vec{x}}^{\mathrm{c/nc}}((A_{i\nu}(I \cap^{\mathrm{d}} P))_{\nu<n_i} \to^{\mathrm{c/nc}} P\vec{t}))_{i<k} \to^{\mathrm{c}} P\vec{x}),$$

where $\forall^{\mathrm{c/nc}}$ and $\to^{\mathrm{c/nc}}$ in the step formulas are decorated as the clause formulas are.

We decorate inductively defined conjunction, disjunction and existential quantifier. There are variants in order to specify which part of the formula has the computational significance.

Example 2.5.3 (Decorated conjunction)**.** We define the following decorated conjunctions. Here, the superscripts d, u, l and r stand for double, uniformed, left and right, respectively.

$$\begin{aligned}
&A \to^{\mathrm{c}} B \to^{\mathrm{c}} A \wedge^{\mathrm{d}} B && (\wedge^{\mathrm{d}})^+\\
&A \wedge^{\mathrm{d}} B \to^{\mathrm{c}} (A \to^{\mathrm{c}} B \to^{\mathrm{c}} P) \to^{\mathrm{c}} P && (\wedge^{\mathrm{d}})^-\\
&A \to^{\mathrm{nc}} B \to^{\mathrm{nc}} A \wedge^{\mathrm{u}} B && (\wedge^{\mathrm{u}})^+\\
&A \wedge^{\mathrm{u}} B \to^{\mathrm{c}} (A \to^{\mathrm{nc}} B \to^{\mathrm{nc}} P) \to^{\mathrm{c}} P && (\wedge^{\mathrm{u}})^-\\
&A \to^{\mathrm{c}} B \to^{\mathrm{nc}} A \wedge^{\mathrm{l}} B && (\wedge^{\mathrm{l}})^+\\
&A \wedge^{\mathrm{l}} B \to^{\mathrm{c}} (A \to^{\mathrm{c}} B \to^{\mathrm{nc}} P) \to^{\mathrm{c}} P && (\wedge^{\mathrm{l}})^-\\
&A \to^{\mathrm{nc}} B \to^{\mathrm{c}} A \wedge^{\mathrm{r}} B && (\wedge^{\mathrm{r}})^+\\
&A \wedge^{\mathrm{r}} B \to^{\mathrm{c}} (A \to^{\mathrm{nc}} B \to^{\mathrm{c}} P) \to^{\mathrm{c}} P && (\wedge^{\mathrm{r}})^-
\end{aligned}$$

For predicates Q_1 and Q_2 of the same arity $(\vec{\tau})$, we define $Q_1 \cap^\circ Q_2$ to be $\{\vec{x} \,|\, Q_1\vec{x} \wedge^\circ Q_2\vec{x}\}$, where $\circ \in \{\mathrm{d}, \mathrm{l}, \mathrm{r}\}$.

Notice that $\cap$ is not used in the definitions of $\wedge^\circ$. Definition 2.5.2 is therefore defined without any circularity. Due to Definition 2.5.12, $\wedge^{\mathrm{u}}$ can be treated as a non-computational inductive predicate.

Example 2.5.4 (Decorated disjunction)**.** We define the following decorated disjunctions.

Here, the superscripts d, u, l and r stand for double, uniformed, left and right, respectively.

$$
\begin{array}{ll}
A \to^{\mathrm{c}} A \vee^{\mathrm{d}} B & (\vee^{\mathrm{d}})^+_0 \\
B \to^{\mathrm{c}} A \vee^{\mathrm{d}} B & (\vee^{\mathrm{d}})^+_1 \\
A \vee^{\mathrm{d}} B \to^{\mathrm{c}} (A \to^{\mathrm{c}} P) \to^{\mathrm{c}} (B \to^{\mathrm{c}} P) \to^{\mathrm{c}} P & (\vee^{\mathrm{d}})^- \\
A \to^{\mathrm{nc}} A \vee^{\mathrm{u}} B & (\vee^{\mathrm{u}})^+_0 \\
B \to^{\mathrm{nc}} A \vee^{\mathrm{u}} B & (\vee^{\mathrm{u}})^+_1 \\
A \vee^{\mathrm{u}} B \to^{\mathrm{c}} (A \to^{\mathrm{nc}} P) \to^{\mathrm{c}} (B \to^{\mathrm{nc}} P) \to^{\mathrm{c}} P & (\vee^{\mathrm{u}})^- \\
A \to^{\mathrm{c}} A \vee^{\mathrm{l}} B & (\vee^{\mathrm{l}})^+_0 \\
B \to^{\mathrm{nc}} A \vee^{\mathrm{l}} B & (\vee^{\mathrm{l}})^+_1 \\
A \vee^{\mathrm{l}} B \to^{\mathrm{c}} (A \to^{\mathrm{c}} P) \to^{\mathrm{c}} (B \to^{\mathrm{nc}} P) \to^{\mathrm{c}} P & (\vee^{\mathrm{l}})^- \\
A \to^{\mathrm{nc}} A \vee^{\mathrm{r}} B & (\vee^{\mathrm{r}})^+_0 \\
B \to^{\mathrm{c}} A \vee^{\mathrm{r}} B & (\vee^{\mathrm{r}})^+_1 \\
A \vee^{\mathrm{r}} B \to^{\mathrm{c}} (A \to^{\mathrm{nc}} P) \to^{\mathrm{c}} (B \to^{\mathrm{c}} P) \to^{\mathrm{c}} P & (\vee^{\mathrm{r}})^-
\end{array}
$$

For predicates Q_1 and Q_2 of the same arity $(\vec{\tau})$, we define $Q_1 \cup^\circ Q_2$ to be $\{\vec{x} \mid Q_1\vec{x} \vee^\circ Q_2\vec{x}\}$, where $\circ \in \{\mathrm{d}, \mathrm{u}, \mathrm{l}, \mathrm{r}\}$.

Example 2.5.5 (Decorated existential quantifier)**.** We define the following decorated existential quantifiers. Here, the superscripts d, u, l and r stand for double, uniformed, left and right, respectively.

$$
\begin{array}{llll}
\forall^{\mathrm{c}}_{x^\rho}(A \to^{\mathrm{c}} \exists^{\mathrm{d}}_x A) & (\exists^{\mathrm{d}})^+ & \forall^{\mathrm{nc}}_{x^\rho}(A \to^{\mathrm{nc}} \exists^{\mathrm{u}}_x A) & (\exists^{\mathrm{u}})^+ \\
\exists^{\mathrm{d}}_x A \to^{\mathrm{c}} \forall^{\mathrm{c}}_{x^\rho}(A \to^{\mathrm{c}} P) \to^{\mathrm{c}} P & (\exists^{\mathrm{d}})^- & \exists^{\mathrm{u}}_x A \to^{\mathrm{c}} \forall^{\mathrm{nc}}_{x^\rho}(A \to^{\mathrm{nc}} P) \to^{\mathrm{c}} P & (\exists^{\mathrm{u}})^- \\
\forall^{\mathrm{c}}_{x^\rho}(A \to^{\mathrm{nc}} \exists^{\mathrm{l}}_x A) & (\exists^{\mathrm{l}})^+ & \forall^{\mathrm{nc}}_{x^\rho}(A \to^{\mathrm{c}} \exists^{\mathrm{r}}_x A) & (\exists^{\mathrm{r}})^+ \\
\exists^{\mathrm{l}}_x A \to^{\mathrm{c}} \forall^{\mathrm{c}}_{x^\rho}(A \to^{\mathrm{nc}} P) \to^{\mathrm{c}} P & (\exists^{\mathrm{l}})^- & \exists^{\mathrm{r}}_x A \to^{\mathrm{c}} \forall^{\mathrm{nc}}_{x^\rho}(A \to^{\mathrm{c}} P) \to^{\mathrm{c}} P & (\exists^{\mathrm{r}})^-
\end{array}
$$

Due to Definition 2.5.12, $\exists^{\mathrm{u}}$ can be treated as a non-computational inductive predicate.

Example 2.5.6. Let A, B and C be formulas. Assume conjunction is right associative. In a decorated formula $A \wedge^{\mathrm{r}} B \wedge^{\mathrm{d}} C$ the formula A is treated as a non-computational formula. The others, B and C are treated as a computational formulas. In a decorated formula $A \wedge^{\mathrm{l}} B \wedge^{\mathrm{l}} C$ the formula A is treated as computational formula. The others are treated as non-computational ones.

Example 2.5.7. Let A and B be formulas and x be a variable. In a formula $\exists^{\mathrm{l}}_x(A)$, the variable x is treated as a computational object, and the formula A is treated as a non-computational formula. In a formula $\exists^{\mathrm{r}}_x(A \vee^{\mathrm{u}} B)$ the formula $A \vee^{\mathrm{u}} B$ is treated as a computational formula. The variable x and the formulas A and B are treated as a non-computational variable and non-computational formulas, respectively.

Definition 2.5.8 (Decorating coinductive predicates)**.** Suppose that a coinductively defined predicate ${}^{\mathrm{co}}I = \nu_X(\forall_{\vec{x}_i}((A_{i\nu}(X))_{\nu<n_i} \to X\vec{t}_i))_{i<k}$ is given. As in Definition 2.3.15, we form a formula $\exists_{\vec{x}_i}(\bigwedge_{\nu<n_i} A_{i\nu}(X) \land \bigwedge \vec{E}_i) =: {}^{\mathrm{co}}K_i(X)$ for each $i < k$ and decorate it. We use $\land^{\mathrm{d}}$, $\land^{\mathrm{l}}$, $\land^{\mathrm{r}}$ and $\land^{\mathrm{u}}$ to specify which $A_{i\nu}(X)$ is computational, and also use $\exists^{\mathrm{d}}$, $\exists^{\mathrm{l}}$, $\exists^{\mathrm{r}}$ and $\exists^{\mathrm{u}}$ to specify which x_i is computational. The following is the rule of decoration:

- $A_{i\nu}$ followed by $\to^{\mathrm{c}}$ is computational.
- $A_{i\nu}$ followed by $\to^{\mathrm{nc}}$ is non-computational.
- x_i which occurs with $\forall^{\mathrm{c}}$ is computational.
- x_i which occurs with $\forall^{\mathrm{nc}}$ is non-computational.

For decorated ${}^{\mathrm{co}}K_i(X)$, we form the disjunction $\bigvee_{i<k} {}^{\mathrm{co}}K_i(X)$ by using $\lor^{\mathrm{d}}$, $\lor^{\mathrm{l}}$, $\lor^{\mathrm{r}}$ and $\lor^{\mathrm{u}}$. If there is neither $\to^{\mathrm{c}}$ nor $\forall^{\mathrm{c}}$ in the i-th clause formula, ${}^{\mathrm{co}}K_i(X)$ is treated as non-computational disjunct. Otherwise it is computational. The following is the decorated closure axiom.

$$\forall^{\mathrm{nc}}_{\vec{y}}({}^{\mathrm{co}}I\vec{y} \to^{\mathrm{c}} \bigvee_{i<k} {}^{\mathrm{co}}K_i({}^{\mathrm{co}}I)).$$

The greatest-fixed-point axioms are decorated as follows.

$$\forall^{\mathrm{nc}}_{\vec{y}}(P\vec{y} \to^{\mathrm{c}} \forall^{\mathrm{nc}}_{\vec{y}}(P\vec{y} \to^{\mathrm{c}} \bigvee_{i<k} {}^{\mathrm{co}}K_i({}^{\mathrm{co}}I \cup^{\mathrm{d}} P)) \to^{\mathrm{c}} {}^{\mathrm{co}}I\vec{y}).$$

There are two kinds of non-computational definitions. The first kind is an arbitrary inductively and coinductively defined predicate which is marked as a non-computational one under the two restrictions; the predicate parameters are *invariant* and the competitor predicate in the elimination axiom is *non-computational*. They are called *non-computational inductively (coinductively) defined predicates*. The second kind is a so-called *special non-computational inductive definition* which is used for an inductive definition with one clause formula involving no recursive premise and containing $\forall^{\mathrm{nc}}$ and $\to^{\mathrm{nc}}$ only.

Definition 2.5.9 (Non-computational inductive and coinductive definitions)**.** We extend predicate forms in the following way

$$\begin{aligned} P ::= & X \mid \{\vec{x} \mid A\} \mid \mu_X(\forall^{\mathrm{c/nc}}_{\vec{x}_i}((A_{i\nu})_{\nu<n_i} \to^{\mathrm{c/nc}} X\vec{t}_i))_{i<k} \mid \nu_X(\forall^{\mathrm{c/nc}}_{\vec{x}_i}((A_{i\nu})_{\nu<n_i} \to^{\mathrm{c/nc}} X\vec{t}_i))_{i<k} \mid \\ & \mu^{\mathrm{nc}}_X(\forall^{\mathrm{c/nc}}_{\vec{x}_i}((A_{i\nu})_{\nu<n_i} \to^{\mathrm{c/nc}} X\vec{t}_i))_{i<k} \mid \nu^{\mathrm{nc}}_X(\forall^{\mathrm{c/nc}}_{\vec{x}_i}((A_{i\nu})_{\nu<n_i} \to^{\mathrm{c/nc}} X\vec{t}_i))_{i<k}, \end{aligned}$$

where $\forall^{\mathrm{c/nc}}$ and $\to^{\mathrm{c/nc}}$ are arbitrarily decorated. Let J be $\mu^{\mathrm{nc}}_X(\forall^{\mathrm{c/nc}}_{\vec{x}_i}((A_{i\nu})_{\nu<n_i} \to^{\mathrm{c/nc}} X\vec{t}_i))_{i<k}$ or $\nu^{\mathrm{nc}}_X(\forall^{\mathrm{c/nc}}_{\vec{x}_i}((A_{i\nu})_{\nu<n_i} \to^{\mathrm{c/nc}} X\vec{t}_i))_{i<k}$. A predicate form J is a predicate if $\mathrm{Pred}(J)$ is derived using the following additional rule for Pred.

$$\frac{\text{for all } Y \in \mathrm{FPV}(A) \setminus \vec{X},\ Y \text{ is invariant}}{\mathrm{NC}_{\vec{X}}(A)}\ ,$$

$$\frac{\text{for all } i<k, \text{ for all } \nu<n_i,\ \mathrm{F}(A_{i\nu}) \text{ and } \mathrm{SP}_X(A_{i\nu}) \text{ and } \mathrm{NC}_X(A_{i\nu}) \quad \mathrm{Inhab}(J)}{\mathrm{Pred}(J)}$$

The notion of being invariant is simultaneously given in Definition 2.5.22. Let I and ${}^{\mathrm{co}}I$ be non-computational. inductive and non-computational coinductive predicates, respectively. The axioms I_i^+, I^-, ${}^{\mathrm{co}}I^-$ and ${}^{\mathrm{co}}I^+$ are given as the same formulas as the computational axioms, but the competitor predicate of I^- and ${}^{\mathrm{co}}I^+$ are restricted to be non-computational.

We take a look at examples.

Example 2.5.10 (Non-computational disjunction)**.** We give a non-computational disjunction $A \vee^{\mathrm{nc}} B := \mu_X^{\mathrm{nc}}(A \to X, B \to X)$. The introduction and elimination axioms are the following.

$$\begin{aligned} &A \to A \vee^{\mathrm{nc}} B && (\vee^{\mathrm{nc}})_0^+ \\ &B \to A \vee^{\mathrm{nc}} B && (\vee^{\mathrm{nc}})_1^+ \\ &A \vee^{\mathrm{nc}} B \to (A \to P) \to (B \to P) \to P && (\vee^{\mathrm{nc}})^- \end{aligned}$$

We also introduce the notion of $\cup^{\mathrm{nc}}$ in the same way as decorated disjunctions.

Example 2.5.11 (Weak disjunction)**.** For formulas A and B, we define $A\tilde{\vee}B$ to be $((A \to \mathbf{F}) \wedge (B \to \mathbf{F})) \to \mathbf{F}$ which we call *weak disjunction*. We can prove $A \vee^{\mathrm{nc}} B \to A\tilde{\vee}B$, but cannot prove $A\tilde{\vee}B \to A \vee^{\mathrm{nc}} B$. Although the conclusion $A \vee^{\mathrm{nc}} B$ is non-computational, one has to make a decision, either the left or the right, in order to show it. The premise $A\tilde{\vee}B$ does not provide such information.

The second kind of non-computational inductive definitions are as follows.

Definition 2.5.12 (One clause special non-computational inductive definitions)**.** A non-computational inductive definition $I_{\vec{Y}} \coloneqq \mu_X^{\mathrm{nc}}(\forall_{\vec{x}}((A_{i\nu})_{\nu<n_i} \to X\vec{t}))_{i<k}$ can be a *one clause special non-computational inductive definition* if the following conditions are satisfied

1. There is only one clause formula, namely, $k = 1$.
2. All implications and universal quantifiers are non-computational.
3. Premises $A_{0\nu}$ for $\nu < n_i$ are of the form $Y_j\vec{s}_j$.

When an inductive predicate satisfies the above condition of one-clause special non-computational inductive predicates, we always use the special non-computational inductive version. We take a look at an example.

Example 2.5.13 (Leibniz equality, non-computational conjunction and existential quantifier)**.** We define the following special non-computational inductive predicates.

$$\begin{aligned} &\mathrm{eqd} := \mu_X^{\mathrm{nc}}(\forall_{x^\alpha}^{\mathrm{nc}}(X\,x\,x)), \\ &\wedge^{\mathrm{u}} := \mu_X^{\mathrm{nc}}(A \to^{\mathrm{nc}} B \to^{\mathrm{nc}} X), \\ &\exists^{\mathrm{u}} := \mu_X^{\mathrm{nc}}(\forall_x^{\mathrm{nc}}(Yx \to^{\mathrm{nc}} X)). \end{aligned}$$

Their introduction and elimination axioms are as follows.

$$\forall^{\mathrm{nc}}_{x^\rho}(x \operatorname{eqd} x) \qquad \mathrm{eqd}^+$$
$$\forall^{\mathrm{nc}}_{x^\rho,y^\rho}(x \operatorname{eqd} y \to^{\mathrm{nc}} \forall^{\mathrm{nc}}_{x^\rho}(Pxx) \to^{\mathrm{c}} Pxy) \qquad \mathrm{eqd}^-$$
$$A \to^{\mathrm{nc}} B \to^{\mathrm{nc}} A \wedge^{\mathrm{u}} B \qquad (\wedge^{\mathrm{u}})^+$$
$$A \wedge^{\mathrm{u}} B \to^{\mathrm{nc}} (A \to^{\mathrm{nc}} B \to^{\mathrm{nc}} P) \to^{\mathrm{c}} P \qquad (\wedge^{\mathrm{u}})^-$$
$$\forall^{\mathrm{nc}}_{x^\rho}(A \to^{\mathrm{nc}} \exists^{\mathrm{u}}_x A) \qquad (\exists^{\mathrm{u}})^+$$
$$\forall^{\mathrm{nc}}_{x^\rho}(\exists^{\mathrm{u}}_x A \to^{\mathrm{nc}} \forall^{\mathrm{nc}}_x(A \to^{\mathrm{nc}} P) \to^{\mathrm{c}} P) \qquad (\exists^{\mathrm{u}})^-$$

For eqd we adopt the infix notation as $x \operatorname{eqd} y$. For predicates Q_1 and Q_2 of the same arity $(\vec{\tau})$, we define $Q_1 \cap^{\mathrm{u}} Q_2$ to be $\{\vec{x} \mid Q_1\vec{x} \wedge^{\mathrm{u}} Q_2\vec{x}\}$.

Example 2.5.14. Consider conjunction of formulas A, B and C. If we are interested in computational content of A only, a formula $A \wedge^{\mathrm{d}} B \wedge^{\mathrm{u}} C$ can instead be decorated as $A \wedge^{\mathrm{l}} B \wedge^{\mathrm{u}} C$.

We define the extended type $\tau(A)$ of a formula A. The extended type $\tau(A)$ is either a type or the *null type* denoted by $\circ$. When $\tau(A)$ is a type, it tells us what the type of an extracted program from a proof of A is. If any proof of the formula A does *not* have computational content, there is nothing to extract from a proof and $\tau(A)$ results in the null type. When no confusion may arise, we simply call $\tau(A)$ a type rather than an extended type. Let ρ be a type, then we extend the use of the arrow type in the following way.

$$(\rho \to \circ) := \circ, \quad (\circ \to \rho) := \rho, \quad (\circ \to \circ) := \circ.$$

The types of formulas and predicates are defined by simultaneous induction on the construction of formulas and predicates.

Definition 2.5.15 (Type of formulas and predicates)**.** Let A be a formula and P be a predicate. Assume type variables $\vec{\alpha}$ associated with $\vec{Y} \subseteq \mathrm{FPV}(A)$ and $\vec{Y} \subseteq \mathrm{FPV}(P)$. We define a type or a null type $\tau(A)$ and $\tau(P)$ by simultaneous induction on the construction of formulas and predicates.

$$\tau(A \to^{\mathrm{c}} B) := \tau(A) \to \tau(B), \qquad \tau(A \to^{\mathrm{nc}} B) := \tau(B),$$
$$\tau(\forall^{\mathrm{c}}_{x^\rho} A) := \rho \to \tau(A), \qquad \tau(\forall^{\mathrm{nc}}_{x^\rho} A) := \tau(A),$$
$$\tau(P\vec{t}) := \tau(P), \qquad \tau(\{\vec{x} \mid B\}) := \tau(B),$$
$$\tau(X) := \xi_X, \qquad \tau(Y_i) := \alpha_i,$$

If $I = \mu_X(\vec{K})$ and ${}^{\mathrm{co}}I = \nu_X(\vec{K})$ are computational inductive and coinductive definitions,

$$\tau(I) := \mu_{\xi_X}(\tau(K_0), \ldots, \tau(K_{n-1})), \qquad \tau({}^{\mathrm{co}}I) := \mu_{\xi_X}(\tau(K_0), \ldots, \tau(K_{n-1})),$$

and else if I and ${}^{\mathrm{co}}I$ are non-computational inductive and coinductive definitions,

$$\tau(I) := \tau({}^{\mathrm{co}}I) := \circ.$$

Especially for an inductively (and coinductively) defined predicate I (or ${}^{\mathrm{co}}I$), $\tau(I)$ (or $\tau({}^{\mathrm{co}}I)$) is referred to by the *associated algebra* of I (or ${}^{\mathrm{co}}I$). Conventionally, we write ι_I for both $\tau(I)$ and $\tau({}^{\mathrm{co}}I)$. A formula A is *computational* or *computationally relevant* if $\tau(A)$ is not null. Otherwise, A is *non-computational*. We abbreviate them as c.r. and n.c., respectively. We use these abbreviations also for predicates.

Now we address the issue of the notion of proofs involving $\forall^{\mathrm{nc}}$ and $\to^{\mathrm{nc}}$. The rules in Definition 2.4.1 are used for $\forall^{\mathrm{c}}$ and $\to^{\mathrm{c}}$ as they are. Consider formulas $\forall^{\mathrm{nc}}_x B$ and $A \to^{\mathrm{nc}} B$. The idea is that x and A can be used in B, but they should *not* make any computational significance. We formalize computational object variables and computational assumption variables in a proof, then restrict introduction rules of $\forall^{\mathrm{nc}}$ and $\to^{\mathrm{nc}}$ in order to be computationally consistent. Strictly speaking, the inference rules are simultaneously defined with CV and CA. The following definition of CV is due to Berger [Ber93].

Definition 2.5.16 (Computational object variables)**.** Let M^A be a derivation. If $\tau(A) = \circ$, $\mathrm{CV}(M) = \emptyset$. Otherwise, $\mathrm{CV}(M)$ is defined by induction on the construction of M.

$$
\begin{aligned}
\mathrm{CV}(c^A) &:= \emptyset \ (c^A \text{ is an axiom}),\\
\mathrm{CV}(u^A) &:= \emptyset,\\
\mathrm{CV}((\lambda_{u^A} M^B)^{A\to^{\mathrm{c}}B}) &:= \mathrm{CV}((\lambda_{u^A} M^B)^{A\to^{\mathrm{nc}}B}) := \mathrm{CV}(M),\\
\mathrm{CV}((M^{A\to^{\mathrm{c}}B} N^A)^B) &:= \mathrm{CV}(M) \cup \mathrm{CV}(N),\\
\mathrm{CV}((M^{A\to^{\mathrm{nc}}B} N^A)^B) &:= \mathrm{CV}(M),\\
\mathrm{CV}((\lambda_x M^A)^{\forall^{\mathrm{c}}_x A}) &:= \mathrm{CV}((\lambda_x M^A)^{\forall^{\mathrm{nc}}_x A}) := \mathrm{CV}(M)\backslash\{x\},\\
\mathrm{CV}((M^{\forall^{\mathrm{c}}_x A(x)} r)^{A(r)}) &:= \mathrm{CV}(M) \cup \mathrm{FV}(r),\\
\mathrm{CV}((M^{\forall^{\mathrm{nc}}_x A(x)} r)^{A(r)}) &:= \mathrm{CV}(M).
\end{aligned}
$$

The following definition of CA is due to Ratiu and Schwichtenberg [RS10].

Definition 2.5.17 (Computational assumption variables)**.** Let M^A be a derivation. If $\tau(A) = \circ$, $\mathrm{CA}(M) = \emptyset$. Otherwise, $\mathrm{CA}(M)$ is defined by induction on the construction of M.

$$
\begin{aligned}
\mathrm{CA}(c^A) &:= \emptyset \ (c^A \text{ is an axiom}),\\
\mathrm{CA}(u^A) &:= \{u\},\\
\mathrm{CA}((\lambda_{u^A} M^B)^{A\to^{\mathrm{c}}B}) &:= \mathrm{CA}((\lambda_{u^A} M^B)^{A\to^{\mathrm{nc}}B}) := \mathrm{CA}(M)\backslash\{u\},\\
\mathrm{CA}((M^{A\to^{\mathrm{c}}B} N^A)^B) &:= \mathrm{CA}(M) \cup \mathrm{CA}(N),\\
\mathrm{CA}((M^{A\to^{\mathrm{nc}}B} N^A)^B) &:= \mathrm{CA}(M),\\
\mathrm{CA}((\lambda_x M^A)^{\forall^{\mathrm{c}}_x A}) &:= \mathrm{CA}((\lambda_x M^A)^{\forall^{\mathrm{nc}}_x A}) := \mathrm{CA}(M),\\
\mathrm{CA}((M^{\forall^{\mathrm{c}}_x A(x)} r)^{A(r)}) &:= \mathrm{CA}((M^{\forall^{\mathrm{nc}}_x A(x)} r)^{A(r)}) := \mathrm{CA}(M).
\end{aligned}
$$

We give the following introduction rules for $\forall^{\mathrm{nc}}$ and $\to^{\mathrm{nc}}$. It is necessary in order to keep the computational consistency of our notion of proofs.

Definition 2.5.18 (Inference rules of $\forall^{\mathrm{nc}}$ and $\to^{\mathrm{nc}}$)**.** Let A, B be variables which range over formulas, M ranges over derivations, u ranges over assumption variables, x ranges over object variables. The elimination rules of $\forall^{\mathrm{nc}}$ and $\to^{\mathrm{nc}}$ are the same as ones of $\forall^-$ and $\to^-$. The introduction rule of $\forall^{\mathrm{nc}}$ is

$$\frac{\begin{array}{c}\vdots\, M\\ A\end{array}}{\forall^{\mathrm{nc}}_x A}\,(\forall^{\mathrm{nc}})^+ \qquad \text{provided} \begin{cases} x \notin \mathrm{CV}(M) \text{ and } x \notin \mathrm{FV}(B) \text{ for} \\ \text{any free assumption variable } u^B \text{ in } M. \end{cases}$$

The introduction rule of $\to^{\mathrm{nc}}$ is

$$\frac{\begin{array}{c}[u^A]\\ \vdots\, M\\ B\end{array}}{A \to^{\mathrm{nc}} B}\,(\to^{\mathrm{nc}})^+ u \qquad \text{provided } u^A \notin \mathrm{CA}(M).$$

The elimination rules of $\forall^{\mathrm{nc}}$ and $\to^{\mathrm{nc}}$ are the same as the ones of $\forall^{\mathrm{c}}$ and $\to^{\mathrm{c}}$.

We introduce the *computational equivalence* between formulas. Based on it we can omit some decoration to simplify the matter.

Definition 2.5.19 (Computationally equivalent formulas)**.** Formulas A and A' is computationally equivalent if one can prove both of $A \to^{\mathrm{c}} A'$ and $A' \to^{\mathrm{c}} A$, and also the identity function is a realizer of them.

It is easy to see that $A \to^{\mathrm{c}} B$ and $A \to^{\mathrm{nc}} B$ are computationally equivalent, and so $\forall^{\mathrm{c}}_x A$ and $\forall^{\mathrm{nc}}_x A$ are, provided A is non-computational. We can omit the decorations and simply write $A \to B$ and $\forall_x A$. If A is n.c. and B is c.r., $A \wedge^{\mathrm{d}} B$ and $A \wedge^{\mathrm{r}} B$ are also computationally equivalent.

2.5.2 Program Extraction

We define the program extraction algorithm which translates a proof into a program. The notion of proofs is as given in Section 2.4 with decoration given in Section 2.5.1. The notion of programs is T^+ given in Section 2.2. We first define the realizability interpretation which is a relation on a term in T^+ and a formula. Based on the realizability interpretation, we secondly define the program extraction algorithm. We define the *empty term* which is a realizer of non-computational formulas.

Definition 2.5.20 (Empty term)**.** Let t be a term. The empty term ε is an extension of terms which satisfy the following:

$$t\varepsilon \coloneqq t, \quad \varepsilon t \coloneqq \varepsilon, \quad \varepsilon\varepsilon \coloneqq \varepsilon.$$

For any formula and predicate A, $A\varepsilon \coloneqq A$.

The following is the realizability interpretation which is given by simultaneous induction on the structure of formulas and predicates in Definition 2.3.5. For a predicate P of arity $(\vec{\tau})$ where the length of $\vec{\tau} \geq 0$, we define another predicate $P^{\mathbf{r}}$ of arity $(\tau(P), \vec{\tau})$.

Definition 2.5.21 (Realizability interpretation)**.** Assume a formula A, a predicate P and for each Y_i of arity $(\vec{\delta}_i)$ a predicate variable Y_i^* of arity $(\alpha_i, \vec{\delta}_i)$ for some type variable α_i. We define the realizability interpretation $A^{\mathbf{r}}$ of A by simultaneous induction on the construction of formulas and predicates.

$$
\begin{array}{ll}
(\forall_x^{\mathrm{c}} A)^{\mathbf{r}} := \{u \,|\, \forall_x(A^{\mathbf{r}}(ux))\}, & (A \to^{\mathrm{c}} B)^{\mathbf{r}} := \{u \,|\, \forall_x(A^{\mathbf{r}}x \to B^{\mathbf{r}}(ux))\}, \\
(\forall_x^{\mathrm{nc}} A)^{\mathbf{r}} := \{u \,|\, \forall_x(A^{\mathbf{r}}u)\}, & (A \to^{\mathrm{nc}} B)^{\mathbf{r}} := \{u \,|\, \forall_x(A^{\mathbf{r}}x \to B^{\mathbf{r}}u)\}, \\
(P\vec{s})^{\mathbf{r}} := \{u \,|\, P^{\mathbf{r}}u\vec{s}\}, & Y_i^{\mathbf{r}} := \{u, \vec{x} \,|\, Y_i^* u\vec{x}\} \quad (Y_i \text{ c.r.}), \\
\{\vec{x} \,|\, A\}^{\mathbf{r}} := \{u, \vec{x} \,|\, A^{\mathbf{r}}u\}, & Y_i^{\mathbf{r}} := Y_i \quad (Y_i \text{ n.c.}).
\end{array}
$$

For computational inductively and coinductively defined predicates,

$$
(J_{\vec{P}})^{\mathbf{r}} := \{u, \vec{x} \,|\, J^{\mathbf{r}}_{\vec{P}^{\mathbf{r}}} u\vec{x}\}, \qquad ({}^{\mathrm{co}}J_{\vec{P}})^{\mathbf{r}} := \{u, \vec{x} \,|\, {}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{P}^{\mathbf{r}}} u\vec{x}\}.
$$

For non-computational inductively and coinductively defined predicates,

$$
(J_{\vec{P}})^{\mathbf{r}} := J_{\vec{P}}, \qquad ({}^{\mathrm{co}}J_{\vec{P}})^{\mathbf{r}} := {}^{\mathrm{co}}J_{\vec{P}}.
$$

For one-clause defined non-computational inductively defined predicates,

$$
(J_{\vec{P}})^{\mathbf{r}} := J_{(\exists^{\mathrm{u}}_x(P_i^{\mathbf{r}}x))_{i<n}}.
$$

We can write $u \ \mathbf{r} \ A_{\vec{P}}$ instead of $A^{\mathbf{r}}_{\vec{P}^{\mathbf{r}}} u$.

If P is a predicate variable, $P^{\mathbf{r}}$ is same as P^* from the above definition. Invariant formulas are defined as follows.

Definition 2.5.22 (Invariant formulas and predicates)**.** A formula A and a predicate P are invariant if $A \leftrightarrow \exists_x(A^{\mathbf{r}}x)$ and $P\vec{t} \leftrightarrow \exists_x(P^{\mathbf{r}}x\vec{t})$ for any $\vec{t}$ are derivable in TCF, respectively.

The inductive and coinductive predicates with the superscript $\mathbf{r}$, namely, $I^{\mathbf{r}}$, are called *witnessing predicates*. For each an inductively (or coinductively) defined predicate J, we define the n.c. predicate $J^{\mathbf{r}}$, the witnessing predicate of J. Suppose that the arity of J is $(\vec{\tau})$, then the arity of $J^{\mathbf{r}}$ is $(\iota_J, \vec{\tau})$. We define $J^{\mathbf{r}}$ as a non-computational inductive (or coinductive) predicate. By means of them, the notion of realizability can be internalized in the language of TCF.

Definition 2.5.23 (Witnessing predicates)**.** Suppose that $I_{\vec{Y}} = \mu_X(\vec{K})$ is an inductive predicate of arity $(\vec{\tau})$, where Y_i is of arity $(\vec{\sigma}_i)$. Let ι be the associated algebra of $I_{\vec{Y}}$. Assume predicate variables $\vec{Y}^*$ such that Y_i^* is of arity $(\tau(Y_i), \vec{\sigma}_i)$. Let C_i be the i-th constructor of ι. Let X^* be a predicate variable of arity $(\iota, \vec{\tau})$ and K_i^* be a formula $C_i \ \mathbf{r} \ K_i$ with replacing each occurrence of $X^{\mathbf{r}}$ by X^*. The witnessing predicate $I^{\mathbf{r}}_{\vec{Y}^{\mathbf{r}}}$ is defined to be an n.c. inductive

predicate $\mu^{\mathrm{nc}}_{X^*}(\vec{K}^*)$ of arity $(\iota,\vec{\tau})$. Suppose that K_i is of the form $\forall_{\vec{x}_i}((A_{i\nu}(X))_{\nu<n_i}\to X\vec{t}_i)$. The i-th introduction axiom of $I^{\mathbf{r}}_{\vec{Y}^{\mathbf{r}}}$ is

$$\forall_{\vec{x}_i,\vec{u}_i}((A^{\mathbf{r}}_{i\nu}(I^{\mathbf{r}}_{\vec{Y}^{\mathbf{r}}})u_{i\nu})_{\nu<n_i}\to I^{\mathbf{r}}_{\vec{Y}^{\mathbf{r}}}(C_i\vec{x}_i\vec{u}_i,\vec{t}_i)) \tag{$I^{\mathbf{r}}$)^+_i}$$

where

1. $u_{i\nu}$ occurs if $A_{i\nu}$ is c.r., and it appears as an argument of C_i if $A_{i\nu}$ is followed by $\to^{\mathrm{c}}$,
2. $x_{ii'}$ with $\forall^{\mathrm{c}}$ appears as an argument of C_i.

Let P be a predicate variable of the same arity of $I^{\mathbf{r}}_{\vec{Y}^{\mathbf{r}}}$. Recall that P has to be instantiated by a non-computational predicate. The elimination axiom of $I^{\mathbf{r}}_{\vec{Y}^{\mathbf{r}}}$ is as follows.

$$\forall_{\vec{x},u}(I^{\mathbf{r}}_{\vec{Y}^{\mathbf{r}}}u\vec{x}\to(\forall_{\vec{x}_i,\vec{u}_i}((A^{\mathbf{r}}_{i\nu}(I^{\mathbf{r}}_{\vec{Y}^{\mathbf{r}}}\cap^{\mathrm{u}}P)u_{i\nu})_{\nu<n_i}\to P(C_i\vec{u}_i,\vec{t}_i)))_{i<k}\to Pu\vec{x}) \tag{$I^{\mathbf{r}}$)$^-$}$$

Recall that $\cap^{\mathrm{u}}$ is defined in Example 2.5.13.

Suppose that ${}^{\mathrm{co}}I_{\vec{Y}}$ is the companion predicate of $I_{\vec{Y}}$ above. The witnessing predicate ${}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{Y}^{\mathbf{r}}}=\nu^{\mathrm{nc}}_{X^*}(\vec{K}^{\mathbf{r}})$ of ${}^{\mathrm{co}}I_{\vec{Y}}$ is an n.c. coinductive predicate of arity $(\iota,\vec{\tau})$. Let $\vec{v}$ be fresh variables of the same length of $\vec{y}$ and k be the length of $\vec{K}^{\mathbf{r}}$. Let $A^*_i(X^*)$ be $\exists^{\mathrm{u}}_{\vec{y}_i,\vec{v}_i}(\bigwedge^{\mathrm{nc}}_{\nu<n_i}(A^{\mathbf{r}}_{i\nu}(X^*)v_{i\nu})\wedge\bigwedge\vec{E}\wedge\mathcal{D}u\,\mathrm{eqd}\,\mathbf{in}^k_i(\mathbf{pair}\,\vec{v}_i))$, where $v_{i\nu}$ is not there if $A_{i\nu}$ is n.c. The closure axiom of ${}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{Y}^{\mathbf{r}}}$ is

$$\forall_{\vec{x},u}({}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{Y}^{\mathbf{r}}}u\vec{x}\to\bigvee^{\mathrm{nc}}_{i<k}A^*_i({}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{Y}^{\mathbf{r}}})), \tag{${}^{\mathrm{co}}I^{\mathbf{r}}$)$^-$}$$

Let P be a non-computational predicate variable of the same arity of ${}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{Y}^{\mathbf{r}}}$. The greatest-fixed-point axiom of ${}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{Y}^{\mathbf{r}}}$ is

$$\forall_{\vec{x},u}(Pu\vec{x}\to\forall_{\vec{x},u}(Pu\vec{x}\to\bigvee^{\mathrm{nc}}_{i<k}A^*_i({}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{Y}^{\mathbf{r}}}\cup^{\mathrm{nc}}P))\to{}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{Y}^{\mathbf{r}}}u\vec{x}), \tag{${}^{\mathrm{co}}I^{\mathbf{r}}$)$^+$}$$

where $\cup^{\mathrm{nc}}$ is given in Example 2.5.10.

We give some examples of witnessing predicates.

Example 2.5.24 (Witnessing predicates of conjunction)**.** Let A and B are predicate variables of arities (α) and (β), respectively. Let variables a be ranging over α, b over β, w over $\alpha\times\beta$. We define the witnessing predicates of the conjunctions, namely, the witnessing versions of Example 2.5.3.

$$\begin{aligned}
&A(\wedge^{\mathrm{d}})^{\mathbf{r}}B:=\mu^{\mathrm{nc}}_X(\forall_{a,b}(Aa\to^{\mathrm{nc}}Bb\to^{\mathrm{nc}}X\langle a,b\rangle)),\\
&A(\wedge^{\mathrm{l}})^{\mathbf{r}}B:=\mu^{\mathrm{nc}}_X(\forall_{a,b}(Aa\to^{\mathrm{nc}}Bb\to^{\mathrm{nc}}Xa)),\\
&A(\wedge^{\mathrm{r}})^{\mathbf{r}}B:=\mu^{\mathrm{nc}}_X(\forall_{a,b}(Aa\to^{\mathrm{nc}}Bb\to^{\mathrm{nc}}Xb)).
\end{aligned}$$

The introduction and elimination axioms are

$$\begin{aligned}
&\forall_{a,b}(Aa\to Bb\to(A(\wedge^{\mathrm{d}})^{\mathbf{r}}B)\langle a,b\rangle) && ((\wedge^{\mathrm{d}})^{\mathbf{r}})^+\\
&\forall_w((A(\wedge^{\mathrm{d}})^{\mathbf{r}}B)w\to\forall_{a,b}(Aa\to Bb\to P\langle a,b\rangle)\to Pw) && ((\wedge^{\mathrm{d}})^{\mathbf{r}})^-\\
&\forall_{a,b}(Aa\to Bb\to(A(\wedge^{\mathrm{l}})^{\mathbf{r}}B)a) && ((\wedge^{\mathrm{l}})^{\mathbf{r}})^+\\
&\forall_a((A(\wedge^{\mathrm{l}})^{\mathbf{r}}B)a\to\forall_{a,b}(Aa\to Bb\to Pa)\to Pa) && ((\wedge^{\mathrm{l}})^{\mathbf{r}})^-\\
&\forall_{a,b}(Aa\to Bb\to(A(\wedge^{\mathrm{r}})^{\mathbf{r}}B)b) && ((\wedge^{\mathrm{r}})^{\mathbf{r}})^+\\
&\forall_b((A(\wedge^{\mathrm{r}})^{\mathbf{r}}B)b\to\forall_{a,b}(Aa\to Bb\to Pb)\to Pb). && ((\wedge^{\mathrm{r}})^{\mathbf{r}})^-
\end{aligned}$$

Example 2.5.25 (Witnessing predicates of disjunction)**.** Let P, A and B be predicate variables of arities $()$, (α) and (β), respectively. We define the witnessing predicates of the decorated disjunctions.

$$
\begin{aligned}
&A(\vee^{\mathrm{d}})^{\mathbf{r}}B \coloneqq \mu^{\mathrm{nc}}_X(\forall_a(A\,a \to X(\mathsf{InL}\,a)), \forall_b(B\,b \to X(\mathsf{InR}\,b))),\\
&A(\vee^{\mathrm{l}})^{\mathbf{r}}P \coloneqq \mu^{\mathrm{nc}}_X(\forall_a(A\,a \to X(\mathsf{JustL}\,a)), P \to X\,\mathsf{NoneR}),\\
&P(\vee^{\mathrm{r}})^{\mathbf{r}}B \coloneqq \mu^{\mathrm{nc}}_X(P \to X\,\mathsf{None}, \forall_b(B\,b \to X(\mathsf{Just}\,b))),\\
&P_0(\vee^{\mathrm{u}})^{\mathbf{r}}P_1 \coloneqq \mu^{\mathrm{nc}}_X(P_0 \to X\,\mathsf{T}, P_1 \to X\,\mathsf{F}).
\end{aligned}
$$

The introduction and elimination axioms are as follows.

$$
\begin{aligned}
&\forall_a(A\,a \to (A(\vee^{\mathrm{d}})^{\mathbf{r}}B)(\mathsf{InL}\,a)) && ((\vee^{\mathrm{d}})^{\mathbf{r}})^+_0\\
&\forall_b(B\,b \to (A(\vee^{\mathrm{d}})^{\mathbf{r}}B)(\mathsf{InR}\,b)) && ((\vee^{\mathrm{d}})^{\mathbf{r}})^+_1\\
&\forall_x((A(\vee^{\mathrm{d}})^{\mathbf{r}}B)x \to \forall_a(A\,a \to Q(\mathsf{InL}\,a)) \to \forall_b(B\,b \to Q(\mathsf{InR}\,b)) \to Q\,x) && ((\vee^{\mathrm{d}})^{\mathbf{r}})^-\\
&\forall_a(A\,a \to (A(\vee^{\mathrm{l}})^{\mathbf{r}}P)(\mathsf{JustL}\,a)) && ((\vee^{\mathrm{l}})^{\mathbf{r}})^+_0\\
&P \to (A(\vee^{\mathrm{l}})^{\mathbf{r}}P)\mathsf{NoneR} && ((\vee^{\mathrm{l}})^{\mathbf{r}})^+_1\\
&\forall_u((A(\vee^{\mathrm{l}})^{\mathbf{r}}P)u \to \forall_a(A\,a \to Q(\mathsf{JustL}\,a)) \to (P \to Q\,\mathsf{NoneR}) \to Q\,u) && ((\vee^{\mathrm{l}})^{\mathbf{r}})^-\\
&P \to (P(\vee^{\mathrm{r}})^{\mathbf{r}}B)\mathsf{None} && ((\vee^{\mathrm{r}})^{\mathbf{r}})^+_0\\
&\forall_b(B\,b \to (P(\vee^{\mathrm{r}})^{\mathbf{r}}B)(\mathsf{Just}\,b)) && ((\vee^{\mathrm{r}})^{\mathbf{r}})^+_1\\
&\forall_v((P(\vee^{\mathrm{r}})^{\mathbf{r}}B)v \to (P \to Q\,\mathsf{None}) \to \forall_b(B\,b \to Q(\mathsf{Just}\,b)) \to Q\,v) && ((\vee^{\mathrm{r}})^{\mathbf{r}})^-\\
&P_0 \to (P_0(\vee^{\mathrm{u}})^{\mathbf{r}}P_1)\mathsf{T} && ((\vee^{\mathrm{u}})^{\mathbf{r}})^+_0\\
&P_1 \to (P_0(\vee^{\mathrm{u}})^{\mathbf{r}}P_1)\mathsf{F} && ((\vee^{\mathrm{u}})^{\mathbf{r}})^+_1\\
&\forall_b((P_0(\vee^{\mathrm{u}})^{\mathbf{r}}P_1)b \to (P_0 \to Q\,\mathsf{T}) \to (P_1 \to Q\,\mathsf{F}) \to Q\,b) && ((\vee^{\mathrm{u}})^{\mathbf{r}})^-
\end{aligned}
$$

We introduce the notation of $(\cap^\circ)^{\mathbf{r}}$ and $(\cup^\circ)^{\mathbf{r}}$ for $\circ \in \{\mathrm{d}, \mathrm{l}, \mathrm{r}, \mathrm{u}\}$ in a similar way as we did for the decorated connectives.

Example 2.5.26 (Witnessing predicates of existential quantifiers)**.** Let x be a variable ranging over α, v ranging over τ, and w ranging over $\alpha \times \tau$. Also let Y be a predicate variable of arity (τ, α). We define $\mathrm{ExD}^{\mathbf{r}}_Y$, $\mathrm{ExL}^{\mathbf{r}}_Y$, and $\mathrm{ExR}^{\mathbf{r}}_Y$ of the arities (α, τ), (α), and (τ), respectively.

$$
\begin{aligned}
&\mathrm{ExD}^{\mathbf{r}}_Y := \mu^{\mathrm{nc}}_X(\forall^{\mathrm{nc}}_{x,v}(Yvx \to^{\mathrm{nc}} X\langle x, v\rangle)),\\
&\mathrm{ExL}^{\mathbf{r}}_Y := \mu^{\mathrm{nc}}_X(\forall^{\mathrm{nc}}_{x,v}(Yvx \to^{\mathrm{nc}} Xx)),\\
&\mathrm{ExR}^{\mathbf{r}}_Y := \mu^{\mathrm{nc}}_X(\forall^{\mathrm{nc}}_{x,v}(Yvx \to^{\mathrm{nc}} Xv)).
\end{aligned}
$$

The introduction and elimination axioms are

$$\begin{array}{ll}
\forall_{x,v}(Yvx \to \mathrm{ExD}^{\mathbf{r}}_Y\langle x,v\rangle) & ((\exists^{\mathrm{d}})^{\mathbf{r}})^+\\
\forall_w(\mathrm{ExD}^{\mathbf{r}}_Y w \to \forall_{x,v}(Yvx \to P\langle x,v\rangle) \to Pw) & ((\exists^{\mathrm{d}})^{\mathbf{r}})^-\\
\forall_{x,v}(Yvx \to \mathrm{ExL}^{\mathbf{r}}_Y x) & ((\exists^{\mathrm{l}})^{\mathbf{r}})^+\\
\forall_x(\mathrm{ExL}^{\mathbf{r}}_Y x \to \forall_{x,v}(Yvx \to Px) \to Px) & ((\exists^{\mathrm{l}})^{\mathbf{r}})^-\\
\forall_{x,v}(Yvx \to \mathrm{ExR}^{\mathbf{r}}_Y v) & ((\exists^{\mathrm{r}})^{\mathbf{r}})^+\\
\forall_v(\mathrm{ExR}^{\mathbf{r}}_Y v \to \forall_{x,v}(Yvx \to Pv) \to Pv). & ((\exists^{\mathrm{r}})^{\mathbf{r}})^-
\end{array}$$

For given formulas $\exists^\circ_x Q$ where $\circ \in \{\mathrm{d},\mathrm{l},\mathrm{r}\}$, we can also write $(\exists^\circ_x)^{\mathbf{r}}Q^{\mathbf{r}}$ for the corresponding witnessing predicates $\mathrm{ExD}^{\mathbf{r}}_{Q^{\mathbf{r}}}$, $\mathrm{ExL}^{\mathbf{r}}_{Q^{\mathbf{r}}}$, and $\mathrm{ExR}^{\mathbf{r}}_{Q^{\mathbf{r}}}$.

We define the term extraction algorithm et based on the realizability interpretation.

Definition 2.5.27 (Program extraction)**.** For a proof M of a formula A we define a term $\mathrm{et}(M)$ or ε of type $\tau(A)$ or $\circ$ by induction on the construction of the proof M. If $\tau(A) = \circ$, $\mathrm{et}(M) \coloneqq \varepsilon$. Otherwise,

$$\begin{array}{ll}
\mathrm{et}(u^A) := x^{\tau(A)}_{u^A} \text{ where } x_{u^A} \text{ is uniquely associated with } u^A, & \\
\mathrm{et}(I^+_i) := C_i, & \mathrm{et}(I^-) := \mathcal{R}^\tau_\iota,\\
\mathrm{et}({}^{\mathrm{co}}I^-) := \mathcal{D}_\iota, & \mathrm{et}({}^{\mathrm{co}}I^+) := {}^{\mathrm{co}}\mathcal{R}^\tau_\iota,\\
\mathrm{et}((\lambda_{u^A}M^B)^{A\to^{\mathrm{c}}B}) := \lambda_{x^{\tau(A)}_u}(\mathrm{et}(M)), & \mathrm{et}((\lambda_{u^A}M^B)^{A\to^{\mathrm{nc}}B}) := \mathrm{et}(M),\\
\mathrm{et}(M^{A\to^{\mathrm{c}}B}N^A) := \mathrm{et}(M)\mathrm{et}(N), & \mathrm{et}(M^{A\to^{\mathrm{nc}}B}N^A) := \mathrm{et}(M),\\
\mathrm{et}((\lambda_{x^\rho}M^A)^{\forall^{\mathrm{c}}_x A}) := \lambda_{x^\rho}\mathrm{et}(M), & \mathrm{et}((\lambda_{x^\rho}M^A)^{\forall^{\mathrm{nc}}_x A}) := \mathrm{et}(M),\\
\mathrm{et}((M^{\forall^{\mathrm{c}}_x A}r)^{A(r)}) := \mathrm{et}(M)r, & \mathrm{et}((M^{\forall^{\mathrm{nc}}_x A}r)^{A(r)}) := \mathrm{et}(M),
\end{array}$$

where I and ${}^{\mathrm{co}}I$ are computational, $\iota := \tau(I) = \tau({}^{\mathrm{co}}I)$ and τ is determined by the type of the competitor predicate of I^- or ${}^{\mathrm{co}}I^+$. For a special non-computational predicate I, $\mathrm{et}(I^-) \coloneqq \mathrm{id}$.

Example 2.5.28 (Computational content of the identity lemma)**.** We can prove a trivial statement $A \to A$ which we call the *identity lemma*. Proving it by assuming A and using it to derive the goal, there is the computational content $\mathtt{Id}_\alpha$ extracted from this proof.

$$\mathtt{Id}_\alpha : \alpha \to \alpha, \qquad \mathtt{Id}_\alpha x \mapsto x.$$

Although this lemma has no logical use to prove a formula, we can use this lemma to fine tune the extracted program by switching block/unblock of the conversion rule. Assume formulas A, B and C and a proof M of A. We prove $(A \to^{\mathrm{c}} B) \to^{\mathrm{c}} (A \to^{\mathrm{c}} C) \to^{\mathrm{c}} B \wedge^{\mathrm{d}} C$. Assume two premises. Asserting A, which is proven, and use the identity lemma to $A \to^{\mathrm{c}} B \wedge^{\mathrm{d}} C$. We assume A and apply the assumptions to derive B and C, then by $(\wedge^{\mathrm{d}})^+$. The extracted program is normalized to be $\lambda_{f,g}(\mathtt{Id}\,\lambda_u\langle fu, gu\rangle\,\mathtt{P})$, where $\mathtt{P}$ is the

normalized $\mathrm{et}(M)$. When we prove straightforwardly without the identity lemma, the extracted program is normalized to be $\lambda_{f,g}\langle f\,\mathtt{P}, g\,\mathtt{P}\rangle$ which involves two times of separated normalization of $\mathrm{et}(M)$. We introduce the notation of *let-construction* for $\mathtt{Id}$ in the following way: $\lambda_{f,g}(\mathbf{let}\ u = \mathtt{P}\ \mathbf{in}\ \langle fu, gu\rangle)$.

A use of this technique is also found in Section 3.4.

2.5.3 Soundness Theorem

In this section we address the issue of the correctness of our program extraction. We prove the *soundness theorem* which claims that if there is a proof M of a formula A, there is a term which provably realizes A obtained by program extraction from M.

Theorem 2.5.29 (Soundness)**.** *Let M be a proof of A from assumptions $u_i^{B_i}$. We have a proof of* $\mathrm{et}(M)$ $\mathbf{r}$ A *from assumptions* x_{u_i} $\mathbf{r}$ B_i.

Proof. By induction on the structure of M.

Case $u : B$. Trivially, $\mathrm{et}(u) = x_u$ $\mathbf{r}$ B.

Case $(\lambda_{u^A} M^B)^{A\to^{\mathrm{c}}B}$. We find a derivation of $\mathrm{et}(\lambda_u M)$ $\mathbf{r}$ $A \to^{\mathrm{c}} B$, which is $\forall_x(x\ \mathbf{r}\ A \to \mathrm{et}(M)[x_u/x]\ \mathbf{r}\ B)$. By induction hypothesis, we have a derivation of $\mathrm{et}(M)$ $\mathbf{r}$ B from x_u $\mathbf{r}$ A, hence by introduction rules of $\to$ and $\forall$.

Case $M^{A\to^{\mathrm{c}}B}N^A$. We find a derivation of $\mathrm{et}(MN)$ $\mathbf{r}$ B, i.e., $\mathrm{et}(M)\mathrm{et}(N)$ $\mathbf{r}$ B. By induction hypothesis we have derivations of $\mathrm{et}(M)$ $\mathbf{r}$ $A \to^{\mathrm{c}} B$, namely, $\forall_x(x\ \mathbf{r}\ A \to \mathrm{et}(M)x\ \mathbf{r}\ B)$ and $\mathrm{et}(N)$ $\mathbf{r}$ A. The goal is derived by elimination rules of $\forall$ and $\to$.

Case $(\lambda_x M^A)^{\forall^{\mathrm{c}}_x A}$. We find a derivation of $\mathrm{et}(\lambda_x M)$ $\mathbf{r}$ $\forall^{\mathrm{c}}_x A$, i.e., $\forall_x(\mathrm{et}(M)\ \mathbf{r}\ A)$. By induction hypothesis we have a derivation of $\mathrm{et}(M)$ $\mathbf{r}$ A, hence the goal by an introduction rule of $\forall$.

Case $M^{\forall^{\mathrm{c}}_x A(x)}t$. We find a derivation of $\mathrm{et}(Mt)$ $\mathbf{r}$ $A(t)$. By induction hypothesis we have a derivation of $\mathrm{et}(M)$ $\mathbf{r}$ $\forall^{\mathrm{c}}_x A(x)$, i.e., $\forall_x(\mathrm{et}(M)x\ \mathbf{r}\ A(x))$. Since the goal is same as $\mathrm{et}(M)t$ $\mathbf{r}$ $A(t)$, we use the elimination rule of $\forall$.

Case $(\lambda_{u^A} M^B)^{A\to^{\mathrm{nc}}B}$. We find a derivation of $\mathrm{et}(\lambda_u M)$ $\mathbf{r}$ $A \to^{\mathrm{nc}} B$, which is $\forall_y(y\ \mathbf{r}\ A \to \mathrm{et}(M)\ \mathbf{r}\ B)$. By induction hypothesis, we have a derivation of $\mathrm{et}(M)$ $\mathbf{r}$ B from y $\mathbf{r}$ A, hence by introduction rules of $\to$ and $\forall$.

Case $M^{A\to^{\mathrm{nc}}B}N^A$. We find a derivation of $\mathrm{et}(MN)$ $\mathbf{r}$ B, i.e., $\mathrm{et}(M)$ $\mathbf{r}$ B. By induction hypothesis we have derivations of $\mathrm{et}(M)$ $\mathbf{r}$ $A \to^{\mathrm{nc}} B$, namely, $\forall_y(y\ \mathbf{r}\ A \to \mathrm{et}(M)\ \mathbf{r}\ B)$ and $\mathrm{et}(N)$ $\mathbf{r}$ A. The goal is derived by elimination rules of $\forall$ and $\to$.

Case $(\lambda_x M^A)^{\forall^{\mathrm{nc}}_x A}$. We find a derivation of $\mathrm{et}(\lambda_x M)$ $\mathbf{r}$ $\forall^{\mathrm{nc}}_x A$, i.e., $\forall_x(\mathrm{et}(M)\ \mathbf{r}\ A)$. By induction hypothesis we have a derivation of $\mathrm{et}(M)$ $\mathbf{r}$ A, hence the goal by an introduction rule of $\forall$.

Case $M^{\forall^{\mathrm{nc}}_x A(x)}t$. We find a derivation of $\mathrm{et}(Mt)$ $\mathbf{r}$ $A(t)$, i.e., $\mathrm{et}(M)$ $\mathbf{r}$ $A(t)$. By induction hypothesis we have a derivation of $\mathrm{et}(M)$ $\mathbf{r}$ $\forall^{\mathrm{nc}}_x A(x)$, i.e., $\forall_x(\mathrm{et}(M)\ \mathbf{r}\ A(x))$, hence the goal by using an elimination rule of $\forall$.

If the last inference is the monotonicity, Lemma 2.3.23, we can use Lemma 2.5.30. Otherwise the last inference is an axiom of an inductively or coinductively defined predicate.

If it is I^+, I^-, ${}^{\mathrm{co}}I^-$ or ${}^{\mathrm{co}}I^+$ of computational predicate, use Lemma 2.5.31. If it is I^+ or I^- of non-computational inductive predicate, use Lemma 2.5.32. If it is I^+ or I^- of special non-computational inductive predicate, use Lemma 2.5.33. □

We start proving the case of constants. First, we consider the map operator which appears in the conversion rule of recursion and corecursion operators. A map operator is in fact a realizer of a monotonicity formula, which is exactly the statement of Lemma 2.3.23.

Lemma 2.5.30 (Soundness: the monotonicity formula)**.** *Let $A_{\vec{X}}$ be a formula, n be a fixed natural number, P_i, Q_i be predicates for $i < n$ where P_i and Q_i are of arity $(\vec{\delta}_i)$, $\sigma_i = \tau(P_i)$ and $\tau_i = \tau(Q_i)$, and $\rho(\vec{\alpha})$ be $\tau(A_{\vec{X}})$ where $\alpha_i = \tau(X_i)$. Suppose that u is a variable ranging over $\rho_{\vec{\alpha}}$, which stands for $\rho(\vec{\alpha})$, v ranging over some of τ_i and w_i ranging over $\tau_i \to \sigma_i$. The map operator $\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\rho(\vec{\alpha})}$ is a realizer of the monotonicity formula, namely,*

$$\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\rho(\vec{\alpha})}\ \mathbf{r}\ \forall_{\vec{x}}(A_{\vec{P}} \to (\forall_{\vec{x}}(P_i\vec{x} \to Q_i\vec{x}))_{i<n} \to A_{\vec{Q}}).$$

Proof. We prove the claim by simultaneous induction on the structure of $A_{\vec{X}}$.

Case $A_{\vec{X}} = \forall_{\tilde{x}}\tilde{A}_{\vec{X}}$. Assume the induction hypothesis with a parameter variable $\tilde{x}$,

$$\forall_{\vec{x},u}(\tilde{A}^{\mathbf{r}}_{\vec{P}^{\mathbf{r}}}u \to \forall_{\vec{w}}((\forall_{\vec{x},v}(P_i^{\mathbf{r}}v\vec{x} \to Q_i^{\mathbf{r}}(w_iv)\vec{x}))_{i<n} \to \tilde{A}^{\mathbf{r}}_{\vec{Q}^{\mathbf{r}}}(\mathcal{M}^{\vec{\tau}\to\vec{\sigma}}_{\lambda_{\vec{\alpha}}\rho_{\vec{\alpha}}}\,u\,\vec{w}))).$$

We prove

$$\forall_{\vec{x},u}(\forall_{\tilde{x}}(\tilde{A}^{\mathbf{r}}_{\vec{P}^{\mathbf{r}}}u) \to \forall_{\vec{w}}((\forall_{\vec{x},v}(P_i^{\mathbf{r}}v\vec{x} \to Q_i^{\mathbf{r}}(w_iv)\vec{x}))_{i<n} \to \forall_{\tilde{x}}(\tilde{A}^{\mathbf{r}}_{\vec{Q}^{\mathbf{r}}}(\mathcal{M}^{\vec{\tau}\to\vec{\sigma}}_{\lambda_{\vec{\alpha}}\rho_{\vec{\alpha}}}\,u\,\vec{w})))).$$

Let $\vec{x}$ and u be given and assume $\forall_{\tilde{x}}(\tilde{A}^{\mathbf{r}}_{\vec{P}^{\mathbf{r}}}u)$, let $\vec{w}$ be given and assume $\forall_{\vec{x},v}(P_i^{\mathbf{r}}v\vec{x} \to Q_i^{\mathbf{r}}(w_iv)\vec{x})$ for each $i < n$, and also let $\tilde{x}$ be given. The assumption $\forall_{\tilde{x}}(\tilde{A}^{\mathbf{r}}_{\vec{P}^{\mathbf{r}}}u)$ implies $\tilde{A}^{\mathbf{r}}_{\vec{P}^{\mathbf{r}}}u$. We prove $\tilde{A}^{\mathbf{r}}_{\vec{Q}^{\mathbf{r}}}(\mathcal{M}^{\vec{\tau}\to\vec{\sigma}}_{\lambda_{\vec{\alpha}}\rho_{\vec{\alpha}}}\,u\,\vec{w})$ by using the i.h. with replacing the free variable $\tilde{x}$ in i.h. as necessary.

Case $A_{\vec{X}} = B \to \tilde{A}_{\vec{X}}$. Assume $\rho_{\vec{\alpha}} = \tau(B \to \tilde{A}_{\vec{X}}) = \pi \to \tilde{\rho}_{\vec{\alpha}}$. Notice that $\vec{\alpha}$, which came from the free occurrences of $\vec{X}$, don't occur in π due to the condition of the formula construction of B. Assume the induction hypothesis

$$\forall_{\vec{x},u}(\tilde{A}^{\mathbf{r}}_{\vec{P}^{\mathbf{r}}}u \to \forall_{\vec{w}}((\forall_{\vec{x},v}(P_i^{\mathbf{r}}v\vec{x} \to Q_i^{\mathbf{r}}(w_iv)\vec{x}))_{i<n} \to \tilde{A}^{\mathbf{r}}_{\vec{Q}^{\mathbf{r}}}(\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\tilde{\rho}_{\vec{\alpha}}}\,u\,\vec{w}))),$$

and we prove

$$\begin{aligned}\forall_{\vec{x},u}(\forall_y(B^{\mathbf{r}}y \to \tilde{A}^{\mathbf{r}}_{\vec{P}^{\mathbf{r}}}(uy)) \to \forall_{\vec{w}}((\forall_{\vec{x},v}(P_i^{\mathbf{r}}v\vec{x} \to Q_i^{\mathbf{r}}(w_iv)\vec{x}))_{i<n} \to \\ \forall_y(B^{\mathbf{r}}y \to \tilde{A}^{\mathbf{r}}_{\vec{Q}^{\mathbf{r}}}(\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}(\pi\to\tilde{\rho}_{\vec{\alpha}})}\,u\,\vec{w}\,y)))).\end{aligned}$$

Let $\vec{x}$ and u be given and assume $\forall_y(B^{\mathbf{r}}y \to \tilde{A}^{\mathbf{r}}_{\vec{P}^{\mathbf{r}}}(uy))$, $\vec{w}$ be given and assume $\forall_{\vec{x},v}(P_i^{\mathbf{r}}v\vec{x} \to Q_i^{\mathbf{r}}(w_iv)\vec{x})$ for each $i < n$, and also let y be given and assume $B^{\mathbf{r}}y$. We prove that $\tilde{A}^{\mathbf{r}}_{\vec{Q}^{\mathbf{r}}}(\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}(\pi\to\tilde{\rho}_{\vec{\alpha}})}u\vec{w}y)$. By conversion, $\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}(\pi\to\tilde{\rho}_{\vec{\alpha}})}u\vec{w}y \mapsto \mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\tilde{\rho}_{\vec{\alpha}}}(uy)\vec{w}$, and also $\tilde{A}^{\mathbf{r}}_{\vec{P}^{\mathbf{r}}}(uy)$ holds, hence the i.h. implies the goal.

Case $A_{\vec{X}} = \tilde{P}\vec{r}$ where $\tilde{P}$ is a predicate. We proceed by induction on $\tilde{P}$.

Side case $\tilde{P}$ is a predicate variable. The conclusion $\tilde{P}^{\mathbf{r}}(\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\rho} u\vec{w})\vec{r}$ is same as the first premise $\tilde{P}^{\mathbf{r}}u\vec{r}$, because $\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\rho} u\vec{w} \mapsto u$.

Side case $\tilde{P} = \{\vec{z}\,|\,\tilde{A}\}$ where $\tilde{A}$ is a formula. Assume the following side induction hypothesis for $\tilde{A}$.

$$\forall_u(\tilde{A}^{\mathbf{r}}_{\vec{P}^{\mathbf{r}}}u \to \forall_{\vec{w}}((\forall_{\vec{x},v}(P_i^{\mathbf{r}}v\vec{x} \to Q_i^{\mathbf{r}}(w_iv)\vec{x}))_{i<n} \to \tilde{A}^{\mathbf{r}}_{\vec{Q}^{\mathbf{r}}}(\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\tilde{\rho}_{\vec{\alpha}}}\, u\, \vec{w}))).$$

Since $\vec{z}$ are parameters of this i.h., the goal holds by replacing variables.

Side case $\tilde{P} = J(\vec{R}_{\vec{X}})$ where $J(\vec{X}')$ is an inductively defined predicate $\mu_Z(\vec{K})$ where $K_i = \forall_{\vec{x}}((\tilde{A}_{i\nu}(\vec{X}', Z))_{\nu<n_i} \to Z\vec{t})$ with some decoration, and $\vec{R}$ is a list of predicates of the same length of $\vec{X}'$. Then, $\rho_{\vec{\alpha}} = \iota_{\vec{\pi}(\vec{\alpha})}$ where $\iota_{\vec{\beta}}$ is defined to be an algebra $\tau(J_{\vec{X}'}) = \mu_\xi(\vec{\kappa})$ where $\kappa_i = (\rho_{i\nu}(\vec{\beta}, \xi))_{\nu<n_i} \to \xi$ and $\pi_i(\vec{\alpha})$ is $\tau(R_i(\vec{X}))$. We prove the following formula

$$\forall_{\vec{x},u}(J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}})}u\vec{x} \to \forall_{\vec{w}}((\forall_{\vec{x},v}(P_i^{\mathbf{r}}v\vec{x} \to Q_i^{\mathbf{r}}(w_iv)\vec{x}))_{i<n} \to J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{Q}^{\mathbf{r}})}(\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\iota(\vec{\pi}(\vec{\alpha}))}\, u\, \vec{w})\vec{x})).$$

Let $\vec{x}$ and u be given and assume $J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}})}u\vec{x}$. Also let $\vec{w}$ of type $\vec{\sigma} \to \vec{\tau}$ be given and assume $\forall_{\vec{x},v}(P_i^{\mathbf{r}}v\vec{x} \to Q_i^{\mathbf{r}}(w_iv)\vec{x})$ for each $i < n$. The goal is $J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{Q}^{\mathbf{r}})}(\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_\alpha\iota(\vec{\pi}(\vec{\alpha}))}\, u\, \vec{w})\vec{x}$, which is abbreviated as $Ru\vec{x}$ for fixed $\vec{w}$. By means of $(J^{\mathbf{r}})^-$ for $J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}})}u\vec{x}$, it suffices to prove each step formula of the form

$$\forall_{\vec{x},\vec{u}}((\tilde{A}^{\mathbf{r}}_{i\nu}(\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}}), J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}})} \cap^{\mathrm{u}} R)u_\nu)_{\nu<n_i} \to R(C_i^{\vec{\pi}(\vec{\sigma})}\vec{x}\vec{u}, \vec{x})),$$

where $C_i^{\vec{\beta}}$ is the corresponding constructor of the algebra $\iota_{\vec{\beta}}$. We let $\vec{x}$ and $\vec{u}$ be given and assume $\tilde{A}^{\mathbf{r}}_{i\nu}(\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}}), J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}})} \cap^{\mathrm{u}} R)u_\nu$ for each $\nu < n_i$. Let $f_i \coloneqq \mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\pi_i(\vec{\alpha})} \cdot \vec{w}$. Applying the conversion rule of $\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_\alpha\iota(\vec{\pi}(\vec{\alpha}))}$, the goal is same as

$$J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{Q}^{\mathbf{r}})}(C_i^{\vec{\pi}(\vec{\tau})}\vec{x}(\mathcal{M}^{\vec{\pi}(\vec{\sigma}),\iota_{\vec{\pi}(\vec{\sigma})}\to\vec{\pi}(\vec{\tau}),\iota_{\vec{\pi}(\vec{\tau})}}_{\lambda_{\vec{\beta},\gamma}\rho_{i\nu}(\vec{\beta},\gamma)} u_\nu\vec{f}(\mathcal{M}^{\vec{\pi}(\vec{\sigma})\to\vec{\pi}(\vec{\tau})}_{\iota} \cdot \vec{f}))_{\nu<n_i}, \vec{x}).$$

It suffices to show $\tilde{A}^{\mathbf{r}}_{i\nu}(\vec{R}^{\mathbf{r}}(\vec{Q}^{\mathbf{r}}), J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{Q}^{\mathbf{r}})})(\mathcal{M}^{\vec{\pi}(\vec{\sigma}),\iota_{\vec{\pi}(\vec{\sigma})}\to\vec{\pi}(\vec{\tau}),\iota_{\vec{\pi}(\vec{\tau})}}_{\lambda_{\vec{\beta},\gamma}\rho_{i\nu}(\vec{\beta},\gamma)} u_\nu\vec{f}(\mathcal{M}^{\vec{\pi}(\vec{\tau})\to\vec{\pi}(\vec{\sigma})}_{\iota} \cdot \vec{f}))$ for each $\nu < n_i$. Since $\tilde{A}_{i\nu}(\vec{X}, Z)$ is a formula of sub construction, we can use the following induction hypothesis to proceed.

$$\begin{aligned}
&\forall_{\vec{x},u}(\tilde{A}^{\mathbf{r}}_{i\nu}(\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}}), J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}})} \cap^{\mathrm{u}} R)u \to \\
&\quad \forall_{\vec{f},g}((\forall_{\vec{x},v}(R_i^{\mathbf{r}}(\vec{P}^{\mathbf{r}})v\vec{x} \to R_i^{\mathbf{r}}(\vec{Q}^{\mathbf{r}})(f_iv, \vec{x})))_{i<n} \to \forall_{\vec{x},v}((J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}})} \cap^{\mathrm{u}} R)v\vec{x} \to J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{Q}^{\mathbf{r}})}(gv, \vec{x})) \\
&\qquad \to \tilde{A}^{\mathbf{r}}_{i\nu}(\vec{R}^{\mathbf{r}}(\vec{Q}^{\mathbf{r}}), J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{Q}^{\mathbf{r}})})(\mathcal{M}^{\vec{\pi}(\vec{\sigma}),\iota_{\vec{\pi}(\vec{\sigma})}\to\vec{\pi}(\vec{\tau}),\iota_{\vec{\pi}(\vec{\tau})}}_{\lambda_{\vec{\beta},\gamma}\rho_{i\nu}(\vec{\beta},\gamma)} u\vec{f}g))).
\end{aligned}$$

We already have the first premise in the assumption. The second ones are straightforward by induction hypotheses $\forall_{\vec{x},v}(R_i^{\mathbf{r}}(\vec{P}^{\mathbf{r}})v\vec{x} \to \forall_{\vec{w}}((\forall_{\vec{x},v}(P_i^{\mathbf{r}}v\vec{x} \to Q_i^{\mathbf{r}}(w_iv, \vec{x})))_{i<n} \to$

$R_i^{\mathbf{r}}(\vec{Q}^{\mathbf{r}})(\mathcal{M}v\vec{w},\vec{x})))$. Let g be $\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda\vec{\alpha}\iota(\vec{\pi}(\vec{\alpha}))}\cdot\vec{w}$, then the last premise is trivial since $Rv\vec{x}$ is same as $J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{Q}^{\mathbf{r}})}(\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda\vec{\alpha}\iota(\vec{\pi}(\vec{\alpha}))}v\vec{w},\vec{x})$.

Side case $\tilde{P}={}^{\mathrm{co}}J(\vec{R}(\vec{P}))$ where ${}^{\mathrm{co}}J(\vec{X}')$ is a coinductively defined predicate $\nu_X(\vec{K})$ where $K_i=\forall_{\vec{x}}((\tilde{A}_{i\nu}(\vec{X}',Z))_{\nu<n_i}\to Z\vec{t})$, and $\vec{R}$ is a list of predicates. Then, $\rho_{\vec{\alpha}}=\iota_{\vec{\pi}(\vec{\alpha})}$ where $\iota_{\vec{\beta}}$ is defined to be an algebra $\tau({}^{\mathrm{co}}J(\vec{X}'))=\mu_\xi(\vec{\kappa})$ where $\kappa_i=(\rho_{i\nu}(\vec{\beta},\xi))_{\nu<n_i}\to\xi$, and $\pi_i(\vec{\alpha})$ is $\tau(R_i(\vec{X}))$. We prove the following formula

$$\forall_{\vec{x},u}({}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}})}u\vec{x}\to\forall_{\vec{w}}((\forall_{\vec{x},u}(P_i^{\mathbf{r}}u\vec{x}\to Q_i^{\mathbf{r}}(w_iu)\vec{x}))_{i<n}\to{}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{Q}^{\mathbf{r}})}(\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda\vec{\alpha}\iota_{\vec{\pi}(\vec{\alpha})}}u\vec{w})\vec{x})).$$

Let $\vec{x}$ and u be given and assume ${}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}})}u\vec{x}$. Also let $\vec{w}$ be given and assume $\forall_{\vec{x},u}(P_i^{\mathbf{r}}u\vec{x}\to Q_i^{\mathbf{r}}(w_iu)\vec{x})$ for each $i<n$. We prove ${}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{Q}^{\mathbf{r}})}(\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda\vec{\alpha}\iota_{\vec{\pi}(\vec{\alpha})}}u\vec{w})\vec{x}$ by using $({}^{\mathrm{co}}J^{\mathbf{r}})^+$ with the competitor predicate $R:=\{u,\vec{x}\,|\,\exists_v(u\operatorname{eqd}\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda\vec{\alpha}\iota_{\vec{\pi}(\vec{\alpha})}}v\,\vec{w}\wedge{}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}})}v\vec{x})\}$. It suffices to prove the costep formula

$$\forall_{\vec{x},u}(Ru\vec{x}\to\bigvee\nolimits^{\mathrm{nc}}_{i<k}(\exists^{\mathrm{u}}_{\vec{y},\vec{v}}(\bigwedge\nolimits_{\nu<n_i}\tilde{A}^{\mathbf{r}}_{i\nu}(\vec{R}^{\mathbf{r}}(\vec{Q}^{\mathbf{r}}),{}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{Q}^{\mathbf{r}})}\cup^{\mathrm{nc}}R)v_\nu\wedge\bigwedge\vec{E}_i\wedge u\operatorname{eqd}C_i^{\vec{\pi}(\vec{\tau})}\vec{v}))).$$

Let $\vec{x}$ and u be given and assume $Ru\vec{x}$, which yields v' such that $u\operatorname{eqd}\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda\vec{\alpha}\iota_{\vec{\pi}(\vec{\alpha})}}v'\,\vec{w}$ and ${}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}})}v'\vec{x}$ hold. Applying $({}^{\mathrm{co}}J^{\mathbf{r}})^-$ to ${}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}})}v'\vec{x}$, the following holds.

$$\bigvee\nolimits^{\mathrm{nc}}_{i<k}\big(\exists^{\mathrm{u}}_{\vec{y},\vec{v}}(\bigwedge\nolimits_{\nu<n_i}\tilde{A}^{\mathbf{r}}_{i\nu}(\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}}),{}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}})})v_\nu\wedge\bigwedge\vec{E}_i\wedge v'\operatorname{eqd}C_i^{\vec{\pi}(\vec{\sigma})}\vec{v})\big).\tag{2.1}$$

We apply $\vee^-$ for (2.1) to the goal, then it suffices to prove the following for each i. Here we used $u\operatorname{eqd}\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda\vec{\alpha}\iota_{\vec{\pi}(\vec{\alpha})}}v'\vec{w}$.

$$\begin{aligned}&\forall_{\vec{x},v'}\big(\exists^{\mathrm{u}}_{\vec{y},\vec{v}}(\bigwedge\nolimits_{\nu<n_i}\tilde{A}^{\mathbf{r}}_{i\nu}(\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}}),{}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}})})v_\nu\wedge\bigwedge\vec{E}_i\wedge v'\operatorname{eqd}C_i^{\vec{\pi}(\vec{\sigma})}\vec{v})\to\\&\quad\bigvee\nolimits^{\mathrm{nc}}_{i<k}(\exists^{\mathrm{u}}_{\vec{y},\vec{v}}(\bigwedge\nolimits_{\nu<n_i}\tilde{A}^{\mathbf{r}}_{i\nu}(\vec{R}^{\mathbf{r}}(\vec{Q}^{\mathbf{r}}),{}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{Q}^{\mathbf{r}})}\cup^{\mathrm{nc}}R)v_\nu\wedge\bigwedge\vec{E}_i\wedge\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda\vec{\alpha}\iota_{\vec{\pi}(\vec{\alpha})}}v'\vec{w}\operatorname{eqd}C_i^{\vec{\pi}(\vec{\tau})}\vec{v}))\big).\end{aligned}$$

Let $\vec{x}$ and v' be given and assume the premise. Using $(\exists^{\mathrm{u}})^-$, there are $\vec{y}$ and $\vec{v}$ which satisfy the premise. Let f_j be $\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda\vec{\alpha}\pi_j(\vec{\alpha})}\cdot\vec{w}$. We use to the goal $\vee_i^+$ and $(\exists^{\mathrm{u}})^+$ with $\vec{y}$ and $\mathcal{M}^{\vec{\pi}(\vec{\sigma})\to\vec{\pi}(\vec{\tau})}_{\lambda_{\vec{\beta},\gamma}\rho_{i\nu}(\vec{\beta},\gamma)}v_\nu\vec{f}(\mathcal{M}^{\vec{\pi}(\vec{\sigma})\to\vec{\pi}(\vec{\tau})}_{\iota}\cdot\vec{f})$ for each $\nu<n_i$. Then, it suffices to prove the following.

$$\begin{aligned}&\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda\vec{\alpha}\iota(\vec{\pi}(\vec{\alpha}))}(C_i^{\vec{\pi}(\vec{\sigma})}\vec{v})\vec{w}\operatorname{eqd}C_i^{\vec{\pi}(\vec{\tau})}(\mathcal{M}^{\vec{\pi}(\vec{\sigma}),\iota_{\vec{\pi}(\vec{\sigma})}\to\vec{\pi}(\vec{\tau}),\iota_{\vec{\pi}(\vec{\tau})}}_{\lambda_{\vec{\beta},\gamma}\rho_{i\nu}(\vec{\beta},\gamma)}v_\nu\vec{f}(\mathcal{M}^{\vec{\pi}(\vec{\sigma})\to\vec{\pi}(\vec{\tau})}_{\iota}\cdot\vec{f}))_{\nu<n_i},\\&\bigwedge_{\nu<n_i}\tilde{A}^{\mathbf{r}}_{i\nu}(\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}}),{}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}})})v_\nu\to\\&\qquad\bigwedge_{\nu<n_i}\tilde{A}^{\mathbf{r}}_{i\nu}(\vec{R}^{\mathbf{r}}(\vec{Q}^{\mathbf{r}}),{}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{Q}^{\mathbf{r}})}\cup^{\mathrm{nc}}R)(\mathcal{M}^{\vec{\pi}(\vec{\sigma}),\iota_{\vec{\pi}(\vec{\sigma})}\to\vec{\pi}(\vec{\tau}),\iota_{\vec{\pi}(\vec{\tau})}}_{\lambda_{\vec{\beta},\gamma}\rho_{i\nu}(\vec{\beta},\gamma)}v_\nu\vec{f}(\mathcal{M}^{\vec{\pi}(\vec{\sigma})\to\vec{\pi}(\vec{\tau})}_{\iota}\cdot\vec{f})).\end{aligned}$$

The former one is apparent by the conversion rule. For the latter one, we end up with proving $\tilde{A}^{\mathbf{r}}_{i\nu}(\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}}),{}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}})})v_\nu\to\tilde{A}^{\mathbf{r}}_{i\nu}(\vec{R}^{\mathbf{r}}(\vec{Q}^{\mathbf{r}}),{}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{Q}^{\mathbf{r}})}\cup^{\mathrm{nc}}R)(\mathcal{M}^{\vec{\pi}(\vec{\sigma}),\iota_{\vec{\pi}(\vec{\sigma})}\to\vec{\pi}(\vec{\tau}),\iota_{\vec{\pi}(\vec{\tau})}}_{\lambda_{\vec{\beta},\gamma}\rho_{i\nu}(\vec{\beta},\gamma)}v_\nu\vec{f}(\mathcal{M}^{\vec{\pi}(\vec{\sigma})\to\vec{\pi}(\vec{\tau})}_{\iota}\cdot$

$\vec{f}$)) for each $\nu < n_i$, by using $\wedge_i^+$ and $\wedge^-$. We use the following induction hypothesis to prove it.

$$\forall_{\vec{x},v}\big(\tilde{A}^{\mathbf{r}}_{i\nu}(\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}}), {}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}})})v \to \forall_{\vec{f}}((\forall_{\vec{x},v}(R^{\mathbf{r}}_i(\vec{P}^{\mathbf{r}})v\vec{x} \to R^{\mathbf{r}}_i(\vec{Q}^{\mathbf{r}})(f_i v)\vec{x}))_{i<n} \to$$
$$\forall_g(\forall_{\vec{x},u}({}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}})}u\vec{x} \to ({}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{Q}^{\mathbf{r}})} \cup^{\mathrm{nc}} R)(gu,\vec{x})) \to \tilde{A}^{\mathbf{r}}_{i\nu}(\vec{R}^{\mathbf{r}}(\vec{Q}^{\mathbf{r}}), {}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{Q}^{\mathbf{r}})} \cup^{\mathrm{nc}} R)(\mathcal{M}_{\rho_{i\nu}} v \vec{f} g))\big).$$

It suffices to prove $\tilde{A}^{\mathbf{r}}_{i\nu}(\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}}), {}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}})})v_\nu$, $\forall_{\vec{x},v}(R^{\mathbf{r}}_i(\vec{P}^{\mathbf{r}})v\vec{x} \to R^{\mathbf{r}}_i(\vec{Q}^{\mathbf{r}})(f_i v)\vec{x})$ for each $i < n$, and $\forall_{\vec{x},u}({}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}})}u\vec{x} \to ({}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{Q}^{\mathbf{r}})} \cup^{\mathrm{nc}} R)(\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\iota(\vec{\pi}(\vec{\alpha}))}u\vec{w},\vec{x}))$. The first one is due to the assumption. The second one comes by another induction hypothesis $\forall_{\vec{x},v}(R^{\mathbf{r}}_i(\vec{P}^{\mathbf{r}})v\vec{x} \to \forall_{\vec{w}}((\forall_{\vec{x},u}(P^{\mathbf{r}}_i u\vec{x} \to Q^{\mathbf{r}}_i(w_i u)\vec{x}))_{i<n} \to R^{\mathbf{r}}_i(\vec{Q}^{\mathbf{r}})(\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\iota(\vec{\pi}(\vec{\alpha}))}v\vec{w})\vec{x}))$. For the last one, let $\vec{x}$ and u be given and assume ${}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}})}u\vec{x}$. Our goal is same as ${}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{Q}^{\mathbf{r}})}(\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\iota(\vec{\pi}(\vec{\alpha}))}u\vec{w},\vec{x}) \vee^{\mathrm{nc}} R(\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\iota(\vec{\pi}(\vec{\alpha}))}u\vec{w},\vec{x})$. We apply $(\vee^{\mathrm{nc}})^+_1$ and prove $R(\mathcal{M}^{\vec{\sigma}\to\vec{\tau}}_{\lambda_{\vec{\alpha}}\iota(\vec{\pi}(\vec{\alpha}))}u\vec{w},\vec{x})$, which is straightforward by the premise ${}^{\mathrm{co}}J^{\mathbf{r}}_{\vec{R}^{\mathbf{r}}(\vec{P}^{\mathbf{r}})}u\vec{x}$. □

We prove the cases of four constants, constructors, recursion operators, destructors and corecursion operators.

Lemma 2.5.31 (Soundness: Computational inductive and coinductive predicates)**.** *We define $I_{\vec{X}}$ of arity $(\vec{\tau})$ to be $\mu_Z(\vec{K})$ where K_i is a formula $\forall^{\mathrm{c/nc}}_{\vec{x}}((A_{i\nu}(\vec{X},Z))_{\nu<n_i} \to^{\mathrm{c/nc}} Z\vec{t})$. Let an algebra $\iota_{\vec{\alpha}}$ be $\tau(I_{\vec{X}})$ where $\alpha_i = \tau(X_i)$.*

1. *For i such that $0 \le i < k$, the introduction axiom I_i^+ is realized by the constructor C_i of the algebra $\iota_{\vec{\alpha}}$.*

2. *The elimination axiom I^- is realized by the recursion operator $\mathcal{R}_{\iota_{\vec{\alpha}}}$.*

We define ${}^{\mathrm{co}}I_{\vec{X}}$ of arity $(\vec{\tau})$ to be $\nu_X(\vec{K})$ where $\vec{K}$ is given as the above. Let ι_α be $\tau({}^{\mathrm{co}}I)$ where $\alpha_i = \tau(X_i)$.

3. *The closure axiom ${}^{\mathrm{co}}I^-$ is realized by the destructor $\mathcal{D}_{\iota_{\vec{\alpha}}}$.*

4. *The greatest-fixed-point axiom ${}^{\mathrm{co}}I^+$ is realized by the corecursion operator ${}^{\mathrm{co}}\mathcal{R}_{\iota_{\vec{\alpha}}}$.*

Proof. (1) We prove $C_i\ \mathbf{r}\ \forall^{\mathrm{c/nc}}_{\vec{x}}((A_{i\nu}(\vec{X},I_{\vec{X}}))_{\nu<n_i} \to^{\mathrm{c/nc}} I_{\vec{X}}\vec{t})$. It unfolds into

$$\forall_{\vec{x},\vec{u}}((A^{\mathbf{r}}_{i\nu}(\vec{X}^{\mathbf{r}}, I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}})u_\nu)_{\nu<n_i} \to I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}(C_i\vec{x}\vec{u},\vec{t}))$$

with the understanding that

1. only those x_j with a computational $\forall^{\mathrm{c}}_{x_j}$ in K_i, and

2. only those u_ν with $A_{i\nu}$ c.r. and followed by $\to^{\mathrm{c}}$ in K_i,

occur as arguments in $C_i\vec{x}\vec{u}$. The goal is same as $(I^{\mathbf{r}})^+_i$ with $\vec{X}^{\mathbf{r}}$ for the parameter predicates.

(2) Let P be a predicate variable of the same arity of $I_{\vec{X}}$, and $\rho := \tau(P)$. We prove $\mathcal{R}^{\rho}_{\iota_{\vec{\alpha}}}\ \mathbf{r}\ \forall_{\vec{x}}(I_{\vec{X}}\vec{x} \to (K_i(I_{\vec{X}}, P))_{i<n} \to P\vec{x})$, where each step formula $K_i(I_{\vec{X}}, P)$ is $\forall_{\vec{x}}((A_{i\nu}(\vec{X}, I_{\vec{X}} \cap^{\mathrm{d}} P))_{\nu<n_i} \to P\vec{t})$ for each i. By unfolding, the goal is same as

$$\forall_{\vec{x},u}(I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}\, u\, \vec{x} \to \forall_{\vec{w}}((K^{\mathbf{r}}_i(I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}, P^{\mathbf{r}})w_i)_{i<k} \to P^{\mathbf{r}}(\mathcal{R}_{\iota_{\vec{\alpha}}}u\vec{w}, \vec{t}))).$$

Let $\vec{x}$ and u be given and assume $I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}u\vec{x}$. Also let $\vec{w}$ be given and assume for each $i < k$,

$$K^{\mathbf{r}}_i(I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}, P^{\mathbf{r}})w_i \tag{2.2}$$

Let $Qu\vec{t}$ be our goal $P^{\mathbf{r}}(\mathcal{R}_{\iota_{\vec{\alpha}}}u\vec{w}, \vec{t})$. By using $(I^{\mathbf{r}})^-$ for $I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}u\vec{x}$ with the competitor predicate Q, it suffices to prove all the step formulas

$$\forall_{\vec{x}}\forall_{\vec{u}}((A^{\mathbf{r}}_{i\nu}(\vec{X}^{\mathbf{r}}, I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}} \cap^{\mathrm{u}} Q)u_\nu)_{\nu<n_i} \to Q(C_i\vec{x}\vec{u}, \vec{t})),$$

with the understanding that only relevant x_j and u_j occur as arguments in $C_i\vec{x}\vec{u}$. Let $\vec{x}$ and $\vec{u}$ be given and assume for each $\nu < n_i$,

$$A^{\mathbf{r}}_{i\nu}(\vec{X}^{\mathbf{r}}, I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}} \cap^{\mathrm{u}} Q)u_\nu. \tag{2.3}$$

We prove $Q(C_i\vec{x}\vec{u}, \vec{t})$, which is same as

$$P^{\mathbf{r}}(w_i\vec{x}(\mathcal{M}^{\iota_{\vec{\alpha}}\to\iota_{\vec{\alpha}}\times\tau}_{\lambda_\beta\rho_{i\nu}(\vec{\alpha},\beta)}\, u_\nu\, \lambda_a\langle a, \mathcal{R}_{\iota_{\vec{\alpha}}}a\vec{w}\rangle)_{\nu<n_i}, \vec{t}) \tag{2.4}$$

where $\rho_{i\nu}(\vec{\alpha}, \beta)$ is defined to be a type $\tau(A_{i\nu}(\vec{X}, Z))$. Note that $(I \cap^{\mathrm{d}} P)^{\mathbf{r}}$ is same as $\{q^{\iota\times\tau}, \vec{x}\,|\,(\{v^\iota \,|\, I^{\mathbf{r}}v\vec{x}\}(\wedge^{\mathrm{d}})^{\mathbf{r}}\{\tilde{v}^\tau \,|\, P^{\mathbf{r}}\tilde{v}\vec{x}\})q\}$, where ι and τ are the types of realizers of I and P, respectively. By unfolding (2.2), we have

$$\forall_{\vec{x},\vec{q}}((A^{\mathbf{r}}_{i\nu}(\vec{X}^{\mathbf{r}}, I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}(\cap^{\mathrm{d}})^{\mathbf{r}}P^{\mathbf{r}})q_\nu)_{\nu<n_i} \to P^{\mathbf{r}}(w_i\vec{x}_i\vec{q}, \vec{t})),$$

which implies our goal (2.4) provided the following holds for all $\nu < n_i$,

$$A^{\mathbf{r}}_{i\nu}(\vec{X}^{\mathbf{r}}, I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}(\cap^{\mathrm{d}})^{\mathbf{r}}P^{\mathbf{r}})(\mathcal{M}^{\iota_{\vec{\alpha}}\to\iota_{\vec{\alpha}}\times\tau}_{\lambda_\beta\rho_{i\nu}(\vec{\alpha},\beta)}\, u_\nu\, \lambda_a\langle a, \mathcal{R}a\vec{w}\rangle).$$

Instantiating Lemma 2.5.30 as

$$\begin{aligned}\forall_{\vec{x},u}(A^{\mathbf{r}}_{i\nu}(\vec{X}^{\mathbf{r}}, I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}} \cap^{\mathrm{u}} Q)u \to \forall_g(\forall_{\vec{x},u}((I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}} \cap^{\mathrm{u}} Q)u\vec{x} \to (I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}(\cap^{\mathrm{d}})^{\mathbf{r}}P^{\mathbf{r}})(gu, \vec{x})) \to \\ A^{\mathbf{r}}_{i\nu}(\vec{X}^{\mathbf{r}}, I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}(\cap^{\mathrm{d}})^{\mathbf{r}}P^{\mathbf{r}})(\mathcal{M}^{\iota_{\vec{\alpha}}\to\iota_{\vec{\alpha}}\times\tau}_{\lambda_\beta\rho_{i\nu}(\vec{\alpha},\beta)}ug))),\end{aligned}$$

we prove the goal. The first premise $A^{\mathbf{r}}_{i\nu}(\vec{X}^{\mathbf{r}}, I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}} \cap^{\mathrm{u}} Q)u_\nu$ comes by the assumption (2.3). We prove $\forall_{\vec{x},u}((I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}} \cap^{\mathrm{u}} Q)u\vec{x} \to (I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}(\cap^{\mathrm{d}})^{\mathbf{r}}P^{\mathbf{r}})((\lambda_a\langle a, \mathcal{R}a\vec{w}\rangle)u, \vec{x}))$. Let $\vec{x}$, u be given and assume $(I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}} \cap^{\mathrm{u}} Q)u\vec{x}$, namely, $I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}u\vec{x} \wedge^{\mathrm{u}} Qu\vec{x}$. Using $(\wedge^{\mathrm{u}})^-$, we let $\vec{x}$, u, assume $I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}u\vec{x}$ and $Qu\vec{x}$ and then prove $(I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}(\cap^{\mathrm{d}})^{\mathbf{r}}P^{\mathbf{r}})((\lambda_a\langle a, \mathcal{R}a\vec{w}\rangle)u, \vec{x})$, namely, $(\{v \,|\, I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}v\vec{x}\}(\wedge^{\mathrm{d}})^{\mathbf{r}}\{\tilde{v} \,|\, P^{\mathbf{r}}\tilde{v}\vec{x}\})\langle u, \mathcal{R}u\vec{w}\rangle$.

By means of $(\wedge^{\mathbf{r}})_0^+$, it suffices to prove $I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}u\vec{x}$ and $P^{\mathbf{r}}(\mathcal{R}u\vec{w},\vec{x})$. The first one is in the assumption, and the second one is same as $Qu\vec{x}$ which is again in the assumption.

(3) We prove $\mathcal{D}_{\iota_{\vec{\alpha}}}\ \mathbf{r}\ \forall_{\vec{x}}({}^{\mathrm{co}}I_{\vec{X}}\vec{x}\to\bigvee_{i<k}(\exists_{\vec{y}}(\bigwedge_{\nu<n_i}A_{i\nu}(\vec{X},{}^{\mathrm{co}}I_{\vec{X}})\wedge^{\mathrm{l}}\bigwedge\vec{E}_i)))$, which unfolds into

$$\forall_{\vec{x},u}\big({}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}u\vec{x}\to(\bigvee\nolimits^{\mathbf{r}}_{i<k}(\exists^{\mathbf{r}}_{\vec{y}}(\bigwedge\nolimits^{\mathbf{r}}_{\nu<n_i}A^{\mathbf{r}}_{i\nu}(\vec{X}^{\mathbf{r}},{}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}})\,(\wedge^{\mathrm{l}})^{\mathbf{r}}\bigwedge\nolimits^{\mathbf{r}}\vec{E}_i)))(\mathcal{D}u)\big).$$

Let $\vec{x}$ and u be given and assume ${}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}u\vec{x}$. Applying $({}^{\mathrm{co}}I^{\mathbf{r}})^-$ to the assumption, the following holds.

$$\bigvee\nolimits^{\mathrm{nc}}_{i<k}\big(\exists^{\mathrm{u}}_{\vec{y},\vec{v}}(\bigwedge\nolimits_{\nu<n_i}A^{\mathbf{r}}_{i\nu}(\vec{X}^{\mathbf{r}},{}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}})v_\nu\wedge\bigwedge\vec{E}_i\wedge u\,\mathrm{eqd}\,C_i\vec{v})\big).$$

Using $(\vee^{\mathrm{nc}})^-$, $(\vee^{\mathbf{r}})_i^+$, $(\exists^{\mathrm{u}})^-$, $(\exists^{\mathbf{r}})^+$, $(\wedge^{\mathrm{u}})^-$ and $(\wedge^{\mathbf{r}})^+$, we end up with proving $A^{\mathbf{r}}_{i\nu}(\vec{X}^{\mathbf{r}},{}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}})v_\nu$ from the very same premise. Here we used $\mathcal{D}(C_i\vec{v})\mapsto\mathbf{in}^k_i(\mathbf{pair}\,\vec{v})$.

(4) We prove

$${}^{\mathrm{co}}\mathcal{R}_{\iota_{\vec{\alpha}}}\ \mathbf{r}\ \forall_{\vec{x}}(P\vec{x}\to\forall_{\vec{x}}(P\vec{x}\to\bigvee_{i<k}(\exists_{\vec{y}_i}(\bigwedge_{\nu<n_i}A_{i\nu}(\vec{X},{}^{\mathrm{co}}I_{\vec{X}}\cup P)\wedge^{\mathrm{l}}\bigwedge\vec{E}_i)))\to{}^{\mathrm{co}}I_{\vec{X}}\vec{x}),$$

which unfolds into

$$\begin{aligned}\forall_{\vec{x},u}(P^{\mathbf{r}}u\vec{x}\to\forall_w(\forall_{\vec{x},u}(P^{\mathbf{r}}u\vec{x}\to(\bigvee\nolimits^{\mathbf{r}}_{i<k}(\exists^{\mathbf{r}}_{\vec{y}_i}(\bigwedge\nolimits^{\mathbf{r}}_{\nu<n_i}A^{\mathbf{r}}_{i\nu}(\vec{X}^{\mathbf{r}},{}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}\cup^{\mathbf{r}}P^{\mathbf{r}})\,(\wedge^{\mathrm{l}})^{\mathbf{r}}\bigwedge\nolimits^{\mathbf{r}}\vec{E}_i)))(wu))\to\\ {}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}({}^{\mathrm{co}}\mathcal{R}_{\iota_{\vec{\alpha}}}uw,\vec{x}))).\end{aligned}$$

Let $\vec{x}$, u be given and assume $P^{\mathbf{r}}u\vec{x}$. Let w be given and assume

$$\forall_{\vec{x},u}(P^{\mathbf{r}}u\vec{x}\to(\bigvee\nolimits^{\mathbf{r}}_{i<k}(\exists^{\mathbf{r}}_{\vec{y}_i}(\bigwedge\nolimits^{\mathbf{r}}_{\nu<n_i}A^{\mathbf{r}}_{i\nu}(\vec{X}^{\mathbf{r}},{}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}\cup^{\mathbf{r}}P^{\mathbf{r}})\,(\wedge^{\mathrm{l}})^{\mathbf{r}}\bigwedge\nolimits^{\mathbf{r}}\vec{E}_i)))(wu)).\tag{2.5}$$

Recall that ${}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}\cup^{\mathbf{r}}P^{\mathbf{r}}$ stands for $\{q^{\iota+\tau},\vec{x}\,|\,(\{v\,|\,{}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}v\vec{x}\}\vee^{\mathbf{r}}\{\tilde{v}\,|\,P^{\mathbf{r}}\tilde{v}\vec{x}\})q\}$. We prove ${}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}({}^{\mathrm{co}}\mathcal{R}_{\iota_{\vec{\alpha}}}uw,\vec{x})$ by using ${}^{\mathrm{co}}I^+$ with a competitor predicate $Q:=\{v,\vec{x}\,|\,\exists_{\tilde{v}}(v\,\mathrm{eqd}\,{}^{\mathrm{co}}\mathcal{R}_{\iota_{\vec{\alpha}}}\tilde{v}w\wedge P^{\mathbf{r}}\tilde{v}\vec{x})\}$. It suffices to prove the following costep formula.

$$\forall_{\vec{x},v}(Qv\vec{x}\to\bigvee\nolimits^{\mathrm{nc}}_{i<k}\exists^{\mathrm{u}}_{\vec{y},\vec{u}}(\bigwedge\nolimits_{\nu<n_i}A^{\mathbf{r}}_{i\nu}(\vec{X}^{\mathbf{r}},{}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}\cup^{\mathrm{nc}}Q)u_\nu\wedge\bigwedge\vec{E}_i\wedge v\,\mathrm{eqd}\,C_i\vec{u})).$$

Let $\vec{x}$, v be given and assume $Qv\vec{x}$, which yields some $\tilde{v}$ such that $v\,\mathrm{eqd}\,{}^{\mathrm{co}}\mathcal{R}_{\iota_{\vec{\alpha}}}\tilde{v}w$ and $P^{\mathbf{r}}\tilde{v}\vec{x}$. By the conversion rule ${}^{\mathrm{co}}\mathcal{R}\tilde{v}w$ goes to

$$\begin{aligned}&\mathrm{Case}\ w\tilde{v}\ \mathrm{of}\,\mathbf{in}^k_0v_0\to C_0(\mathcal{M}^{\tau\to\tau+\iota}_{\lambda_\beta\rho_{0\nu}(\vec{\alpha},\beta)}(\pi^{n_0}_\nu v_0)[\mathrm{id},{}^{\mathrm{co}}\mathcal{R}\cdot w])_{\nu<n_0}\\ &\qquad\vdots\\ &\mathbf{in}^k_{k-1}v_{k-1}\to C_{k-1}(\mathcal{M}^{\tau\to\tau+\iota}_{\lambda_\beta\rho_{(k-1)\nu}(\vec{\alpha},\beta)}(\pi^{n_{k-1}}_\nu v_{k-1})[\mathrm{id},{}^{\mathrm{co}}\mathcal{R}\cdot w])_{\nu<n_{k-1}}.\end{aligned}$$

Let $t(w\tilde{u})$ be the above term. Applying (2.5) to the assumption $P^{\mathbf{r}}\tilde{u}\vec{x}$,

$$(\bigvee\nolimits^{\mathbf{r}}_{i<k}(\exists^{\mathbf{r}}_{\vec{y}}(\bigwedge\nolimits^{\mathbf{r}}_{\nu<n_i}A^{\mathbf{r}}_{i\nu}(\vec{X}^{\mathbf{r}},{}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}\cup^{\mathbf{r}}P^{\mathbf{r}}))\,(\wedge^{\mathrm{l}})^{\mathbf{r}}\bigwedge\nolimits^{\mathbf{r}}\vec{E}_i))(w\tilde{u})$$

holds. Using $(\vee^{\mathbf{r}})^-$ for this formula to the following goal

$$\bigvee_{i<k}^{\mathrm{nc}} \exists^{\mathrm{u}}_{\vec{y},\vec{u}}(\bigwedge_{\nu<n_i} A^{\mathbf{r}}_{i\nu}(\vec{X}^{\mathbf{r}}, {}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}} \cup^{\mathrm{nc}} Q)u_\nu \wedge \bigwedge \vec{E}_i \wedge t(w\tilde{u})\,\mathrm{eqd}\, C_i\vec{u}),$$

it suffices to prove for each $i < k$ that the following holds

$$\begin{aligned}&\forall_q((\exists^{\mathbf{r}}_{\vec{y}}(\bigwedge^{\mathbf{r}}_{\nu<n_i} A^{\mathbf{r}}_{i\nu}(\vec{X}^{\mathbf{r}}, {}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}} \cup^{\mathbf{r}} P^{\mathbf{r}})\,(\wedge^{\mathrm{l}})^{\mathbf{r}}\ \bigwedge \vec{E}_i))q \to \\ &\quad \bigvee_{i<k}^{\mathrm{nc}} \exists^{\mathrm{u}}_{\vec{y},\vec{u}}(\bigwedge^{\mathrm{nc}}_{\nu<n_i} A^{\mathbf{r}}_{i\nu}(\vec{X}^{\mathbf{r}}, {}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}} \cup^{\mathrm{nc}} Q)u_\nu \wedge \bigwedge \vec{E}_i \wedge t(\mathbf{in}^k_i q)\,\mathrm{eqd}\, C_i\vec{u})).\end{aligned}$$

Let q be given and assume the premise. We use $(\vee^{\mathrm{nc}})^+_i$ to the goal. Using $(\exists^{\mathbf{r}})^-$, $(\exists^{\mathrm{u}})^+$ and $(\wedge^{\mathbf{r}})^-$, it suffices to prove $\forall_{\vec{x},\vec{y}}(\bigwedge \vec{E}_i \to \forall^{\mathrm{nc}}_{\vec{s}}((A^{\mathbf{r}}_{i\nu}(\vec{X}^{\mathbf{r}}, {}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}} \cup^{\mathbf{r}} P^{\mathbf{r}})s_\nu)_{\nu<n_i} \to \exists^{\mathrm{u}}_{\vec{u}}(\bigwedge_{\nu<n_i} A^{\mathbf{r}}_{i\nu}(\vec{X}^{\mathbf{r}}, {}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}} \cup^{\mathrm{nc}} Q)u_\nu \wedge \bigwedge \vec{E}_i \wedge t(\mathbf{in}^k_i(\mathbf{pair}\vec{s}))\,\mathrm{eqd}\, C_i\vec{u})))$. By the conversion rule,

$$t(\mathbf{in}^k_i(\mathbf{pair}\vec{s})) \mapsto C_i(\mathcal{M}s_0[\mathrm{id}, {}^{\mathrm{co}}\mathcal{R}\cdot w]) \ldots (\mathcal{M}s_{n_i-1}[\mathrm{id}, {}^{\mathrm{co}}\mathcal{R}\cdot w]).$$

Using $(\exists^{\mathrm{u}})^+$ with $u_\nu := \mathcal{M}s_\nu[\mathrm{id}, {}^{\mathrm{co}}\mathcal{R}\cdot w]$ and also $\wedge^+$, we end up with proving $A^{\mathbf{r}}_{i\nu}(\vec{X}^{\mathbf{r}}, {}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}} \cup^{\mathrm{nc}} Q)(\mathcal{M}s_\nu[\mathrm{id}, {}^{\mathrm{co}}\mathcal{R}\cdot w])$. By Lemma 2.5.30 it suffices to prove $\forall_{\vec{x},q^{\iota+\tau}}(({}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}} \cup^{\mathbf{r}} P^{\mathbf{r}})^{\mathbf{r}} q\vec{x} \to ({}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}} \cup^{\mathrm{nc}} Q)([\mathrm{id}, {}^{\mathrm{co}}\mathcal{R}\cdot w]q, \vec{x}))$. Let $\vec{x}$, q be given and assume $({}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}} \cup^{\mathbf{r}} P^{\mathbf{r}})q\vec{x}$, namely, $(\{v \mid {}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}} v\vec{x}\} \vee^{\mathbf{r}} \{\tilde{v} \mid P^{\mathbf{r}}\tilde{v}\vec{x}\})q$. We use $(\vee^{\mathbf{r}})^-$. From ${}^{\mathrm{co}}I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}} v\vec{x}$ and $q = \mathsf{InL}\, v$, the goal is straightforward. From $P^{\mathbf{r}}\tilde{v}\vec{x}$ and $q = \mathsf{InR}\, \tilde{v}$, the right disjunct, which is same as $Q({}^{\mathrm{co}}\mathcal{R}\tilde{v}w, \vec{x})$, is implied from the premise. □

We also address the soundness result for non-computational predicates.

Lemma 2.5.32 (Soundness: non-computational inductive and coinductive predicates)**.** *Let $I_{\vec{Y}}$ be a non-computational inductive predicate defined to be $\mu^{\mathrm{nc}}_Z(\vec{K})$ and an algebra $\iota_{\vec{\alpha}}$ to be $\tau(I_{\vec{X}})$ where $\alpha_i = \tau(X_i)$.*

1. *The introduction axiom I^+_i is realized by the empty term ε.*
2. *The elimination axiom I^- is realized by ε.*

Let ${}^{\mathrm{co}}I_{\vec{X}}$ be a non-computational coinductive predicate defined to be $\nu^{\mathrm{nc}}_Z(\vec{K})$ and an algebra $\iota_{\vec{\alpha}}$ to be $\tau({}^{\mathrm{co}}I_{\vec{X}})$ where each α_i is $\tau(X_i)$.

3. *The closure axiom ${}^{\mathrm{co}}I^-$ is realized by the empty term ε.*
4. *The greatest-fixed-point axiom ${}^{\mathrm{co}}I^+$ is realized by the empty term ε.*

Proof. (1) We prove $\varepsilon\ \mathbf{r}\ \forall_{\vec{x}}((A_{i\nu}(I_{\vec{X}}))_{\nu<n_i} \to I_{\vec{X}}\vec{t})$. By unfolding and also by the definition of ε, our goal is same as $\forall_{\vec{x}}\forall_{\vec{u}}((A^{\mathbf{r}}_{i\nu}(I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}})u_\nu)_{\nu<n_i} \to I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}(\varepsilon\vec{x}\vec{u}, \vec{t}))$. Let $\vec{x}$, $\vec{u}$ be given and assume $A^{\mathbf{r}}_{i\nu}(I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}})u_\nu$ for each $\nu < n_i$. The formula $I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}(\varepsilon\vec{x}\vec{u}, \vec{t})$ is same as $I_{\vec{X}}\vec{t}$, because the formula $I_{\vec{X}}\vec{t}$ is non-computational. We use I^+_i, then it suffices to prove $A_{i\nu}(I_{\vec{X}})$ for each $\nu < n_i$. Since $A_{i\nu}(I_{\vec{X}})$ is invariant by the definition of non-computational inductive predicates, the assumption $A^{\mathbf{r}}_{i\nu}(I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}})u_\nu$ yields $A_{i\nu}(I_{\vec{X}})$.

(2) Let P be a predicate variable of the same arity of $I_{\vec{X}}$. We prove $\varepsilon\ \mathbf{r}\ \forall_{\vec{x}}(I_{\vec{X}}\vec{x} \to (K_i(I_{\vec{X}}, P))_{i<k} \to P\vec{x})$ where $K_i(I_{\vec{X}}, P)$ is given by $\forall_{\vec{x}}((A_{i\nu}(\vec{X}, I_{\vec{X}} \cap P))_{\nu<n_i} \to P\vec{t})$. By unfolding the proposition, it is same as

$$\forall_{\vec{x},u}(I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}(u,\vec{x}) \to \forall_{\vec{w}}((K^{\mathbf{r}}_i(I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}, P^{\mathbf{r}})w_i)_{i<k} \to P^{\mathbf{r}}(\varepsilon u\vec{w}, \vec{x}))).$$

Assume $\vec{x}$, u, $I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}(u,\vec{x})$, $\vec{w}$, and $(K^{\mathbf{r}}_i(I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}, P^{\mathbf{r}})w_i)_{i<k}$. We prove $P^{\mathbf{r}}(\varepsilon u\vec{w}, \vec{x})$, which is same as $P\vec{x}$ because P is non-computational. Using I^- with the competitor $\{\vec{x} \mid P\vec{x}\}$, it suffices to prove $I_{\vec{X}}\vec{x}$ and step formulas $(K_i(I_{\vec{X}}, P))_{i<k}$. Since $I_{\vec{X}}$ is n.c., the assumption $I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}(u,\vec{x})$ is same as $I_{\vec{X}}\vec{x}$. The assumption $(K^{\mathbf{r}}_i(I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}}, P^{\mathbf{r}})w_i)_{i<k}$ is same as $\forall_{\vec{x},\vec{u}}((A^{\mathbf{r}}_{i\nu}(\vec{X}^{\mathbf{r}}, I^{\mathbf{r}}_{\vec{X}^{\mathbf{r}}} \cap^{\mathbf{r}} P^{\mathbf{r}})u_\nu)_{\nu<n_i} \to P^{\mathbf{r}}(w_i\vec{u}, \vec{t}))$. This formula is same as the goal, because $A_{i\nu}(\vec{X}, I_{\vec{X}} \cap P)$ for each i and P are all invariant.

(3) We prove $\varepsilon\ \mathbf{r}\ \forall_{\vec{x}}({}^{\mathrm{co}}I_{\vec{X}}\vec{x} \to \bigvee^{\mathrm{nc}}_{i<k}\exists^{\mathrm{u}}_{\vec{y}}(\bigwedge_{\nu<n_i}A_{i\nu}({}^{\mathrm{co}}I_{\vec{X}}) \wedge \bigwedge \vec{E}_i))$. Since ${}^{\mathrm{co}}I_{\vec{X}}\vec{x}$ and the disjunction $\bigvee^{\mathrm{nc}}$ are non-computational, the goal is same as ${}^{\mathrm{co}}I^-$.

(4) We probe

$$\varepsilon\ \mathbf{r}\ \forall_{\vec{x}}(Q\vec{x} \to \forall_{\vec{x}}(Q\vec{x} \to \bigvee^{\mathrm{nc}}_{i<k}\exists^{\mathrm{u}}_{\vec{y}}(\bigwedge_{\nu<n_i}A_{i\nu}({}^{\mathrm{co}}I_{\vec{X}} \cup^{\mathrm{nc}} Q) \wedge \bigwedge \vec{E}_i)) \to {}^{\mathrm{co}}I_{\vec{X}}\vec{x}).$$

Recall that Q, the disjunction $\bigvee^{\mathrm{nc}}$ and ${}^{\mathrm{co}}I$ are non-computational. Then by unfolding, the goal is same as ${}^{\mathrm{co}}I^-$. □

Finally, we prove the soundness results of special non-computational inductively defined predicates.

Lemma 2.5.33 (Soundness: special non-computational inductive predicates)**.** *Let $I_{\vec{Y}}$ be a special non-computational inductive predicate $\mu^{\mathrm{nc}}_X(\forall^{\mathrm{nc}}_{\vec{x}}((A_\nu)_{\nu<n} \to^{\mathrm{nc}} X\vec{t}))$ where A_ν is of the form $Y_j\vec{s}_i$. The empty term ε realizes I^+ and the identity* id *realizes I^-.*

Proof. We prove $\varepsilon\ \mathbf{r}\ \forall^{\mathrm{nc}}_{\vec{x}}((A_\nu)_{\nu<n} \to^{\mathrm{nc}} I_{\vec{Y}}\vec{t})$. By the definition, this formula is same as $\forall_{\vec{x},\vec{u}}((A^{\mathbf{r}}_\nu u_\nu)_{\nu<n} \to I_{\{\vec{z} \mid \exists^{\mathrm{u}}_v(Y_i^{\mathbf{r}} v\vec{z})\}_{i<l}}\vec{t})$. Let $\vec{x}$ and $\vec{u}$ be given and assume $(A^{\mathbf{r}}_\nu u_\nu)_{\nu<n}$. We use I^+ to prove $I_{\{\vec{z} \mid \exists^{\mathrm{u}}_v(Y_i^{\mathbf{r}} v\vec{z})\}_{i<l}}\vec{x})$. It suffices to prove $(\exists^{\mathrm{u}}_v(A^{\mathbf{r}}_\nu v))_{\nu<n}$ which is from the assumption, since A_ν is of the form $Y_i\vec{s}$. We prove id $\mathbf{r}\ \forall^{\mathrm{nc}}_{\vec{x}}(I_{\vec{Y}}\vec{x} \to \forall^{\mathrm{nc}}_{\vec{x}}((A_\nu)_{\nu<n} \to^{\mathrm{nc}} P\vec{t}) \to^{\mathrm{c}} P\vec{x})$. This formula is same as $\forall_{\vec{x},\vec{u}}(I_{\{\vec{z}_i \mid \exists^{\mathrm{u}}_v(Y_i^{\mathbf{r}} v\vec{z}_i)\}_{i<l}}\vec{x} \to \forall_w(\forall_{\vec{x},\vec{u}}((A^{\mathbf{r}}_\nu u_\nu)_{\nu<n} \to P^{\mathbf{r}}w\vec{t}) \to^{\mathrm{c}} P^{\mathbf{r}}w\vec{x})))$. Let $\vec{x}$ and $\vec{u}$ be given and assume $I_{\{\vec{z}_i \mid \exists^{\mathrm{u}}_v(Y_i^{\mathbf{r}} v\vec{z}_i)\}_{i<l}}\vec{x}$, and let w be given and assume $\forall^{\mathrm{nc}}_{\vec{x},\vec{u}}((A^{\mathbf{r}}_\nu u_\nu)_{\nu<n} \to^{\mathrm{nc}} P^{\mathbf{r}}w\vec{t})$. We use I^- for $I_{\{\vec{z}_i \mid \exists^{\mathrm{u}}_v(Y_i^{\mathbf{r}} v\vec{z}_i)\}_{i<l}}\vec{x}$ to prove the goal $Q\vec{x} \coloneqq P^{\mathbf{r}}w\vec{x}$. It suffices to prove the step formula $\forall^{\mathrm{nc}}_{\vec{x}}((\exists^{\mathrm{u}}_v(A^{\mathbf{r}}_\nu v))_{\nu<n} \to^{\mathrm{nc}} Q\vec{t})$. Let $\vec{x}$ be given and assume $(\exists^{\mathrm{u}}_v(A^{\mathbf{r}}_\nu v))_{\nu<n}$ which yields $\vec{v}$ such that $A^{\mathbf{r}}_\nu v_\nu$ for each $\nu < n$, hence the conclusion comes by the assumption. □

Realizability statement $t\ \mathbf{r}\ A$ can be viewed as a correctness assertion of t with respect to a specification A, namely, t is correct with respect to A. The soundness theorem tells us how to construct such a correctness proof from a proof of A. In TCF, this correctness is indeed provable inside of TCF. Informal computational content from the soundness proof offers an algorithm to generate a correctness proof of program extraction.

2.6 Notes

We describe related work and possible future work to conclude this chapter.

2.6.1 T^+ and Partial Continuous Functionals

Our notion of computability is based on the notion of partial continuous functionals, which provides the denotational semantics of the formal calculus T^+. Within the current formulation of T^+, it is not possible to talk about approximation of partial continuous functionals. Huber, Karadais and Schwichtenberg [HKS10] study the theory TCF^+ in order to make the notion of approximation explicit so that it is possible to talk about approximation in the object language. Petrakis [Pet13] has formalized Scott's information systems within TCF^+ and Minlog. One motivation is a formalization of a constructive proof of Kreisel's density theorem, which seems to be feasible if the base theory is as strong as TCF^+.

2.6.2 Non-Computational Part of Proofs

The notion of non-computational part of proofs has been used by Goad [Goa80]. In the field of proof theory, Berger studies the distinction between computational and non-computational ingredients of proofs via computational and non-computational universal quantifiers [Ber93]. Ratiu and Schwichtenberg study the same distinction in implications [RS10]. Such logical connectives are further studied by Benl, Berger, Schwichtenberg, Seisenberger and Zuber [BBS+98], Berger [Ber05] and Schwichtenberg and Wainer [SW12]. Our formulation follows the one by Schwichtenberg and Wainer [SW12]. Paulin-Mohring [PM89] studies a realizability for the Calculus of Construction, considering a distinction between informative and non-informative propositions, which is used to ignore non-computational parts of proofs during program extraction. Based on such a distinction, Letouzey [Let02] studies an extraction mechanism for the Coq proof assistant whose type system is the Calculus of Construction.

2.6.3 Inductive and Coinductive Definitions

Tatsuta studies theories of program extraction **TID** [Tat91] and ν**TID** [Tat98] for inductive and coinductive definitions, respectively. While inductive and coinductive definitions in TCF are by means of strictly positive formulas, inductive and coinductive definitions in **TID** and ν**TID** are by monotone formulas which are not necessarily strictly positive. The target language of the program extraction of **TID** and ν**TID** is untyped. Berger [Ber09, Ber11] studies program extraction from inductive and coinductive definitions. Essentially our formal system is similar to Berger's one. The difference is mainly in the way of formalization.

2.6.4 Normalizability

Normalizability of TCF proofs involving nested inductive coinductive predicates is left as a future work. The computational content of such a normalization proof is expected to be a proof normalizer as studied by Berger [Ber93] and by Berger, Berghofer, Letouzey and Schwichtenberg [BBLS06]. Although corecursion operators in T^+ are not meant to be terminating, normalizability of nested recursion operators should be proven. Abel and Altenkirch gave a predicative proof of a lambda calculus with nested inductive and coinductive types [AA99]. Differently from our case, their corecursion is designed to be terminating as Hagino [Hag87b, Hag87a].

2.6.5 Program Extraction in Other Proof Assistants

There are implementations of proof assistants which support program extraction. Letouzey recently study program extraction based on realizability in the proof assistant Coq [Let08]. The extraction machinery of Coq can extract from proofs programs in OCaml. It is different from one of Minlog in the sense that extracted programs in OCaml are outside of the formal system of Coq, i.e., Calculus of Construction, while T^+ of Minlog is inside of TCF. This machinery of Minlog comes as an advantage, because the realizability relation is still inside the system. It results in an automated program certification via the soundness proof in Minlog. This feature is currently available for proofs involving at most non-nested inductive definitions. Berghofer [Ber02] studies program extraction in Isabelle. This is based on the work by Berghofer and Nipkow [BN02] considering the execution of proof terms in HOL which is the logical foundation of Isabelle. There are more proof assistants which uses the Curry-Howard correspondence to execute proof objects. The Nuprl [BC85, Nup] is the first proof assistant which makes use of the proofs-as-programs paradigm. There are other formalization works in program extraction in exact real arithmetic. One is due to Chuang [Chu11] who studies computational content of proofs in Martin-Löf's type theory by using Agda. Proof terms in Martin-Löf's type theory require a special care for non-computational ingredients which come from "postulates" in contrast to the cases of Coq and Minlog.

Chapter 3

Program Extraction in Exact Real Arithmetic

This chapter describes concrete applications of TCF, defined in Chapter 2, to exact real number computation. All case studies in this chapter are not only written in this text but also available as running examples in Minlog [Min]. Formalization is strictly done and proofs and extracted programs are accessible and executable. The material in this chapter is heavily based on works by Berger [Ber09, Ber11] and Berger and Seisenberger [BS10, BS12]. Precisely speaking, the program `cauchysds` is in [Ber09], the program `ave` is in [BS10, BS12] and the programs `ucf1to0`, `ucf0to1`, `app`, `cmp`, `int` are in [Ber09, Ber11].

The main contribution in this chapter is strict formalization of non-trivial case studies in exact real arithmetic and the machine-extraction of programs on the computer. The latter one makes a major difference from the former works. In order to illustrate our work in a readable way, we are going to be informal rather than to show strictly formalized results in this text. In Section A.2 we provide a detailed explanation of the Minlog formalization of the material in Section 3.3 to complement the informality of this chapter. The Minlog package, which is downloadable at `http://www.minlog-system.de/`, contains the implemented version of this chapter. Among other examples, the directory `examples/analysis/` of the Minlog package contains the files organized as follows: Section 3.2 is in `ratsds.scm`, Section 3.3 is in `cauchysds.scm`, Section 3.4 is in `average.scm`, Section 3.5, Section 3.6, Section 3.7, Section 3.8 are in `readwrite.scm`.

To conclude the introduction, we give a road map for our development. There are two dimensions in our present work. One is the extraction of algorithms in exact real arithmetic, and the other is the representation of objects in exact real arithmetic. Our material consists of real numbers and uniformly continuous functions in concrete representations. Each of them appear in two ways, namely, type-1 and type-0. The type-1 representation of real numbers and uniformly continuous functions are of function type. We review basics of exact real arithmetic of them in Section 3.1. In contrast to it, the type-0 representation of real numbers and uniformly continuous functions are of ground type, namely, of non-function types. Such ground-type representations are available via cototal ideals, which we have seen in Chapter 2.

3.1 Basics of Exact Real Arithmetic

In this section we illustrate exact real arithmetic, restricting to relevant material concerning our case studies in this chapter. For a comprehensive account of this field, consult the standard textbooks on constructive analysis by Bishop [Bis67] and by Bishop and Bridges [BB85]. A treatment of Schwichtenberg [Sch06a, Sch06b, Sch12] is also relevant to study computational content from proofs via realizability interpretation.

An aim to study real arithmetic in a constructive setting is to clarify the algorithmic basis of arguments in real arithmetic. We describe an example to find a difficulty to see an algorithmic ground in the classical setting. Assume we have natural numbers, integers and rational numbers. A Cauchy real is defined to be a sequence of rational numbers $(a_n)_n$ satisfying the Cauchyness condition: $\forall_k \exists_l \forall_{n,m \geq l}(|a_n - a_m| \leq 2^{-k})$. In this classical construction of Cauchy reals, one cannot compute a concrete l from an arbitrary fixed k in general. We start with constructivising the definition of Cauchy reals, such that the index l of the convergence is practically computable. In this section, k ranges over integers and n, m range over natural numbers.

3.1.1 Real Numbers

We give a constructive definition of Cauchy reals. A problem in the classical definition of Cauchy reals is that we cannot effectively find an index to ensure the convergence of a rational sequence. We can solve this by providing an additional information to compute such an index.

Definition 3.1.1 (Cauchy reals)**.** A pair of a sequence of rational numbers $(a_n)_n$ and a Cauchy modulus $M : \mathbb{Z} \to \mathbb{Q}$ is a Cauchy real if the following condition is satisfied.

$$\forall_k \forall_{n,m \geq M(k)}(|\alpha_n - \alpha_m| \leq 2^{-k}).$$

We consider two kinds of inequalities, $x \leq y$ and $x < y$ for Cauchy reals x, y. We define nonnegative Cauchy reals and positive Cauchy reals.

Definition 3.1.2 (Nonnegative and positive Cauchy reals)**.** Let $x \coloneqq \langle (a_n)_n, M \rangle$ be a Cauchy real. Define $x \in \mathbb{R}^{0+}$ to be that $-2^{-k} \leq a_{M(k)}$ for all $k \in \mathbb{Z}$. If $x \in \mathbb{R}^{0+}$, x is *nonnegative*. Also define $x \in_k \mathbb{R}^+$ to be that $2^{-k} \leq a_{M(k_1)}$. If $x \in_k \mathbb{R}^+$, x is k-positive. We can omit k when there is no confusion.

By the following lemma, we can find an upper bound of a real number.

Lemma 3.1.3 (Bound of reals)**.** *For every Cauchy real $x \coloneqq \langle (a_n)_n, M \rangle$ we can find an upper bound 2^{k_x} on the elements of the Cauchy sequence: $|a_n| \leq 2^{k_x}$ for all n.*

Proof. Let k_x be $\max\{|a_n| \mid n \leq M(0)\} + 1 \leq 2^{k_x}$. Then, $|a_n| \leq 2^{k_x}$ holds for all n. □

We use an informal lambda notation to denote an anonymous function. We can write $\lambda\!\!\lambda_x M$ instead of a named function f defined by $f(x) \coloneqq M$. We use the notation k_x as given in Lemma 3.1.3.

Definition 3.1.4 (Arithmetic functions)**.** For Cauchy reals $x \coloneqq \langle (a_n)_n, M\rangle$ and $y \coloneqq \langle (b_n)_n, N\rangle$, we define $x+y$, $-x$, $|x|$, $x \cdot y$ and x^{-1} (provided that $|x| \in_l \mathbb{R}^+$) as follows.

$$
\begin{aligned}
&x + y \coloneqq \langle (a_n + b_n)_n, \Bbbk_k \max\big(M(k+1), N(k+1)\big)\rangle,\\
&-x \coloneqq \langle (-a_n)_n, M\rangle,\\
&|x| \coloneqq \langle |a_n|, M\rangle,\\
&x \cdot y \coloneqq \langle (a_n \cdot b_n)_n, \Bbbk_k \max\big(M(k+1+k_{|y|}), N(k+1+k_{|x|})\big)\rangle,\\
&x^{-1} \coloneqq \langle c_n, \Bbbk_k M(2(l+1)+k)\rangle, \quad c_n \coloneqq \begin{cases} (a_n)^{-1} & \text{if } a_n \neq 0\\ 0 & \text{if } a_n = 0. \end{cases}
\end{aligned}
$$

Results of the above defined functions are indeed Cauchy reals.

Definition 3.1.5 (Inequalities of Cauchy reals)**.** Let x, y be Cauchy reals. Define $x \leq y$ by $y - x \in \mathbb{R}^{0+}$ and $x <_k y$ by $y - x \in_k \mathbb{R}^+$. We can omit k when there is no confusion.

An important consequence in constructive real numbers is that we can not compare two reals as in the classical setting. Concretely speaking, this means that we cannot decide $x \leq y$ or $y \leq x$ for arbitrarily given x and y. However, it is possible to compare a real with an interval [Bis67, BB85]. We give this result as the approximate split property.

Lemma 3.1.6 (Approximate split property)**.** *Let x, y and z be given. If $x < y$ then either $x \leq z$ or $z \leq y$.*

Proof. See [Bis67, BB85]. □

Remark 3.1.7. The algorithmic ground of the above lemma is a decision procedure which can be extracted as an executable program from the proof of the lemma if we formalize them in a suitable framework.

3.1.2 Uniformly Continuous Functions

We can give a constructive definition of uniformly continuous functions on a Cauchy real in the same spirit. The application of a uniformly continuous function to a Cauchy real and the composition of uniformly continuous functions are also given.

Definition 3.1.8 (Uniformly continuous functions)**.** We define a uniformly continuous function $f : I \to \mathbb{R}$, where I is a rational interval. Let h be a function of $I \cap \mathbb{Q} \to \mathbb{N} \to \mathbb{Q}$, α be a function of $\mathbb{Z} \to \mathbb{N}$, ω be a function of $\mathbb{Z} \to \mathbb{N}$ and μ and ν be of $\mathbb{Q}$. A tuple $\langle h, \alpha, \omega, \mu, \nu\rangle$ defines a uniformly continuous function f if the following conditions are satisfied.

$$
\begin{aligned}
&\forall_{a,k}\forall_{n,m\geq\alpha(k)}(|h\,a\,n - h\,a\,m| \leq 2^{-k}),\\
&\forall_{a,b,k}\forall_{n\geq\alpha(k)}(|a-b| < 2^{-\omega(k)+1} \to |h\,a\,n - h\,b\,n| \leq 2^{-k}),\\
&\forall_{a,n}(\mu \leq h\,a\,n \leq \nu).
\end{aligned}
$$

We call h a rational mapping, α a uniform Cauchy modulus, ω a modulus of uniform continuity and μ, ν the lower and the upper bound of h, respectively. For a uniformly continuous function f, we let h_f, α_f, ω_f, μ_f and ν_f denote the the above five components, respectively.

We define application of a uniformly continuous function to a Cauchy real.

Definition 3.1.9 (Application of a uniformly continuous function)**.** Let f be a uniformly continuous function and $x \coloneqq \langle (a_n)_n, M\rangle$ be a Cauchy real. Application of f to x, written $f(x)$, is defined to be

$$\langle (h_f(a_n, n))_n, \lambda_k \max\big(\alpha_f(k+2), M(\omega_f(k+1)-1)\big)\rangle.$$

The application $f(x)$ is indeed a Cauchy real.

Lemma 3.1.10 (Application $f(x)$ is a Cauchy real)**.** *Under the assumptions of the above definition, $f(x)$ is a Cauchy real.*

We define composition of uniformly continuous functions.

Definition 3.1.11 (Composition of uniformly continuous functions)**.** Let I, J be rational intervals and $f : I \to \mathbb{R}$, $g : J \to \mathbb{R}$ be uniformly continuous functions. Assume $\nu_f, \mu_f \in J$. The composition $g \circ f$ is defined as follows.

$$\begin{aligned}
&h_{g\circ f} : (I \cap \mathbb{Q}) \to \mathbb{N} \to \mathbb{Q}, \quad h_{g\circ f} \coloneqq \lambda_{a,n} h_g(h_f(a, n), n),\\
&\alpha_{g\circ f} \coloneqq \lambda_k \max(\alpha_g(k+2), \alpha_f(\omega_g(k+1)-1)),\\
&\omega_{g\circ f} \coloneqq \lambda_k \omega_f(\omega_g(k)-1)+1, \quad \mu_{g\circ f} \coloneqq \mu_g, \quad \nu_{g\circ f} \coloneqq \nu_g.
\end{aligned}$$

The above composition is indeed a uniformly continuous function.

Lemma 3.1.12 (Composition $g \circ f$ is a uniformly continuous function)**.** *Under the assumption of the above definition, the composition $g \circ f$ is indeed a uniformly continuous function.*

3.2 Rational Number into Signed Digit Stream

The goal of this section is to extract from proofs programs which translate a rational number into a signed digit stream, SDS in short. The extracted program takes a rational number in $[-1, 1]$ as an input, and computes an SDS representation of the input rational number. An SDS is a possibly non-well founded list of -1, 0 and 1. Suppose that d_i ranges over $\{-1, 0, 1\}$. An SDS $d_0{::}d_1{::}\ldots{::}d_{k-1}{::}\ldots$ approximates a real number as precisely as required via rational intervals

$$\left[\sum_{i=0}^{n-1} \frac{d_i}{2^{i+1}} - \frac{1}{2^n}, \sum_{i=0}^{n-1} \frac{d_i}{2^{i+1}} + \frac{1}{2^n}\right]. \tag{3.1}$$

The SDS representation of real numbers is also referred to by the type-0 representation of real numbers, because signed digit streams are of ground type. A finite list $d_0::d_1::\ldots::d_{k-1}::[]$ means a rational number $\sum_{i=0}^{k-1} \frac{d_i}{2^{i+1}}$, which is located at the center of the k-th interval.

We refer to the intervals $[-1,0]$, $[-\frac{1}{2},\frac{1}{2}]$ and $[0,1]$ by *basic intervals*, and denote them by $\mathbb{I}_{\mathsf{L}}$, $\mathbb{I}_{\mathsf{M}}$ and $\mathbb{I}_{\mathsf{R}}$, respectively. The algorithm in the extracted program is based on the procedure determining for a given real number $x \in [-1,1]$, written $x \in \mathbb{I}$, which of the basic intervals contains x. Repeatedly applying this procedure to a given rational number a we obtain an SDS representation of a.

3.2.1 Definitions

We start by defining algebras for our case study in rational numbers.

Definition 3.2.1 (Numbers)**.** Algebras of positive numbers, integers and rational numbers are defined as follows.

$$\mathbf{P} := \mu_\xi(1^\xi, \mathsf{S}_0^{\xi\to\xi}, \mathsf{S}_1^{\xi\to\xi}), \quad \mathbf{Z} := \mu_\xi(0^\xi, \mathsf{P}^{\mathbf{P}\to\xi}, \mathsf{N}^{\mathbf{P}\to\xi}), \quad \mathbf{Q} := \mu_\xi(\#^{\mathbf{P}\to\mathbf{Z}\to\xi}).$$

The constructor $\#$ is an infix operator so that we write $i\#p$ for $p^{\mathbf{P}}$ and $i^{\mathbf{Z}}$. We assume that the rational arithmetic is already given.

Definition 3.2.2 (Totality of $\mathbf{P}$, $\mathbf{Z}$ and $\mathbf{Q}$)**.** We define the totality predicates $T_{\mathbf{P}}$ of positive numbers, $T_{\mathbf{Z}}$ of integers and $T_{\mathbf{Q}}$ of rational numbers. The introduction and elimination axioms are as follows.

$$T_{\mathbf{P}}\,1, \qquad (T_{\mathbf{P}})_0^+ \qquad \forall_p^{\mathrm{nc}}(T_{\mathbf{P}}\,p \to T_{\mathbf{P}}(\mathsf{S}_0\,p)) \qquad (T_{\mathbf{P}})_1^+$$
$$\forall_p^{\mathrm{nc}}(T_{\mathbf{P}}\,p \to T_{\mathbf{P}}(\mathsf{S}_1\,p)) \qquad (T_{\mathbf{P}})_2^+$$

$$\forall_p^{\mathrm{nc}}(T_{\mathbf{P}}\,p \to P\,1 \to \forall_p^{\mathrm{nc}}(T_{\mathbf{P}}\,p \to P\,p \to P(\mathsf{S}_0\,p)) \to \forall_p^{\mathrm{nc}}(T_{\mathbf{P}}\,p \to P\,p \to P(\mathsf{S}_1\,p)) \to P\,p) \qquad (T_{\mathbf{P}})^-$$

$$\forall_p^{\mathrm{nc}}(T_{\mathbf{P}}\,p \to T_{\mathbf{Z}}(\mathsf{P}p)) \qquad (T_{\mathbf{Z}})_0^+ \qquad T_{\mathbf{Z}}\,0 \qquad (T_{\mathbf{Z}})_1^+$$
$$\forall_p^{\mathrm{nc}}(T_{\mathbf{P}}\,p \to T_{\mathbf{Z}}(\mathsf{N}p)) \qquad (T_{\mathbf{Z}})_2^+$$

$$\forall_i^{\mathrm{nc}}(T_{\mathbf{Z}}\,i \to \forall_p^{\mathrm{nc}}(T_{\mathbf{P}}\,p \to P(\mathsf{P}p)) \to P\,0 \to \forall_p^{\mathrm{nc}}(T_{\mathbf{P}}\,p \to P(\mathsf{N}p)) \to Pi) \qquad (T_{\mathbf{Z}})^-$$

$$\forall_{p,i}^{\mathrm{nc}}(T_{\mathbf{P}}\,p \to T_{\mathbf{Z}}\,i \to T_{\mathbf{Q}}(p\#i)), \qquad (T_{\mathbf{Q}})^+$$
$$\forall_q^{\mathrm{nc}}(T_{\mathbf{Q}}\,q \to \forall_{p,i}^{\mathrm{nc}}(T_{\mathbf{P}}\,p \to T_{\mathbf{Z}}\,i \to P(p\#i)) \to P\,q) \qquad (T_{\mathbf{Q}})^-$$

The following algebra is used to represent SDSs.

Definition 3.2.3 (Interval algebra $\mathbf{I}$)**.** We define the interval algebra $\mathbf{I}$ to be

$$\mu_\xi(\mathsf{I}^\xi, \mathsf{C}_{\mathsf{L}}{}^{\xi\to\xi}, \mathsf{C}_{\mathsf{M}}{}^{\xi\to\xi}, \mathsf{C}_{\mathsf{R}}{}^{\xi\to\xi}).$$

Cototal ideals of the algebra **I** represent SDSs. We are interested in rational numbers in $[-1,1]$. For this restriction, we define a comprehension term Q of arity $(\mathbf{Q})$ to mean total ideals of rational numbers in $[-1,1]$.

Definition 3.2.4 (Rational numbers in $[-1,1]$)**.** We define a comprehension term Q to be $\{a \mid T_{\mathbf{Q}} a \wedge^{\mathrm{l}} a \in \mathbb{I}\}$.

We define an inductive predicate I and a coinductive predicate ${}^{\mathrm{co}}I$ of arity $(\mathbf{Q})$. Finite continued fractions of the specified form are in I. The companion predicate ${}^{\mathrm{co}}I$ contains even non-well founded continued fractions which are real numbers.

Definition 3.2.5 (I and ${}^{\mathrm{co}}I$)**.** We inductively define a predicate I of arity $(\mathbf{Q})$. Let P be a predicate variable of arity $(\mathbf{Q})$. The introduction and elimination axioms are as follows.

$$I0 \qquad I_0^+ \qquad \forall_a^{\mathrm{nc}}(Ia \to I(\tfrac{a-1}{2})) \qquad I_1^+$$
$$\forall_a^{\mathrm{nc}}(Ia \to I(\tfrac{a}{2})) \qquad I_2^+ \qquad \forall_a^{\mathrm{nc}}(Ia \to I(\tfrac{a+1}{2})) \qquad I_3^+$$

$$\begin{aligned}\forall_a^{\mathrm{nc}}(Ia \to{} & P0 \to \forall_a^{\mathrm{nc}}(Ia \to Pa \to P(\tfrac{a-1}{2})) \to \\ & \forall_a^{\mathrm{nc}}(Ia \to Pa \to P(\tfrac{a}{2})) \to \forall_a^{\mathrm{nc}}(Ia \to Pa \to P(\tfrac{a+1}{2})) \to \\ & Pa).\end{aligned} \qquad I^-$$

We coinductively define ${}^{\mathrm{co}}I$, the companion predicate of I. The closure and greatest-fixed-point axioms are as follows.

$$\begin{aligned}\forall_a^{\mathrm{nc}}({}^{\mathrm{co}}Ia \to{} & a \operatorname{eqd} 0 \vee^{\mathrm{r}} \exists_{a_0}^{\mathrm{r}}({}^{\mathrm{co}}Ia_0 \wedge^{\mathrm{l}} a \operatorname{eqd} \tfrac{a_0-1}{2}) \vee^{\mathrm{d}} \\ & \exists_{a_0}^{\mathrm{r}}({}^{\mathrm{co}}Ia_0 \wedge^{\mathrm{l}} a \operatorname{eqd} \tfrac{a_0}{2}) \vee^{\mathrm{d}} \exists_{a_0}^{\mathrm{r}}({}^{\mathrm{co}}Ia_0 \wedge^{\mathrm{l}} a \operatorname{eqd} \tfrac{a_0+1}{2})).\end{aligned} \qquad {}^{\mathrm{co}}I^-$$

$$\begin{aligned}\forall_a^{\mathrm{nc}}(Pa \to{} & \forall_a^{\mathrm{nc}}(Pa \to a \operatorname{eqd} 0 \vee^{\mathrm{r}} \exists_{a_0}^{\mathrm{r}}(({}^{\mathrm{co}}Ia_0 \vee^{\mathrm{d}} Pa_0) \wedge^{\mathrm{l}} a \operatorname{eqd} \tfrac{a_0-1}{2}) \vee^{\mathrm{d}} \\ & \exists_{a_0}^{\mathrm{r}}(({}^{\mathrm{co}}Ia_0 \vee^{\mathrm{d}} Pa_0) \wedge^{\mathrm{l}} a \operatorname{eqd} \tfrac{a_0}{2}) \vee^{\mathrm{d}} \\ & \exists_{a_0}^{\mathrm{r}}(({}^{\mathrm{co}}Ia_0 \vee^{\mathrm{d}} Pa_0) \wedge^{\mathrm{l}} a \operatorname{eqd} \tfrac{a_0}{2})) \to \\ & {}^{\mathrm{co}}Ia),\end{aligned} \qquad {}^{\mathrm{co}}I^+$$

The associated algebra of I and ${}^{\mathrm{co}}I$ is **I**. Note that a list of signed digits, namely, an ideal of **I**, suffices to represent a construction of our fractions. Some rational numbers cannot be finitely represented by our fraction. This is the reason that we make use of cototal ideals in this case study.

3.2.2 Proofs

From a proof of the Proposition 3.2.6 we extract a program which translates a rational number into an SDS. The proof depends on Lemma 3.2.9 below which determines where the given rational number is located.

Proposition 3.2.6 (A rational number to an SDS)**.** $\forall_a^{\mathrm{nc}}(Qa \to {}^{\mathrm{co}}Ia)$.

Proof. Let a be given and assume Qa. Prove ${}^{\mathrm{co}}Ia$ by ${}^{\mathrm{co}}I^{+}$.

$$\begin{aligned}\forall^{\mathrm{nc}}_{a}({Qa} \to \forall^{\mathrm{nc}}_{a}({Qa} \to a \operatorname{eqd} 0 \vee^{\mathrm{r}} \exists^{\mathrm{r}}_{a_0}(({}^{\mathrm{co}}Ia_0 \vee^{\mathrm{d}} Qa_0) \wedge^{\mathrm{l}} a \operatorname{eqd} \tfrac{a_0-1}{2}) \vee^{\mathrm{d}} \\ \exists^{\mathrm{r}}_{a_0}(({}^{\mathrm{co}}Ia_0 \vee^{\mathrm{d}} Qa_0) \wedge^{\mathrm{l}} a \operatorname{eqd} \tfrac{a_0}{2}) \vee^{\mathrm{d}} \\ \exists^{\mathrm{r}}_{a_0}(({}^{\mathrm{co}}Ia_0 \vee^{\mathrm{d}} Qa_0) \wedge^{\mathrm{l}} a \operatorname{eqd} \tfrac{a_0+1}{2})) \to \\ {}^{\mathrm{co}}Ia)\end{aligned}$$

It suffices to prove the second premise. Let a be given and assume Qa. By Lemma 3.2.9, we can determine whether $a \in \mathbb{I}_{\mathsf{L}}$, $a \in \mathbb{I}_{\mathsf{M}}$ or $a \in \mathbb{I}_{\mathsf{R}}$. If $a \in \mathbb{I}_{\mathsf{L}}$, let a_0 be $2a+1$. If $a \in \mathbb{I}_{\mathsf{M}}$, let a_0 be $2a$. If $a \in \mathbb{I}_{\mathsf{L}}$, let a_0 be $2a-1$. In each case the equality and Qa_0 hold. □

The above proof depends on the next lemmas which are proved by considering the construction of rational numbers.

Lemma 3.2.7 (Above 0 or below 0)**.** $\forall^{\mathrm{nc}}_{a}(T_{\mathbf{Q}}a \to a \le 0 \vee 0 \le a)$.

Proof. Assume a and $T_{\mathbf{Q}}a$. Using $(T_{\mathbf{Q}})^{-}$ for $T_{\mathbf{Q}}a$, it suffices to prove

$$\forall^{\mathrm{nc}}_{i^{\mathbf{Z}},p^{\mathbf{P}}}(T_{\mathbf{Z}}i \to T_{\mathbf{P}}p \to (i\#p) \le 0 \vee 0 \le (i\#p)).$$

Let i and p be given and assume $T_{\mathbf{Z}}i$ and $T_{\mathbf{P}}p$. We use $(T_{\mathbf{Z}})^{-}$ for $T_{\mathbf{Z}}i$ to prove $i\#p \le 0 \vee 0 \le i\#p$. There are three step cases. *Case* $i = \mathsf{O}$. $i\#p$ is 0 in $\mathbf{Q}$, hence the goal is trivial. *Case* $i = \mathsf{P}p'$. $i > 0$, hence $0 \le \mathsf{P}p'\#p$. *Case* $i = \mathsf{N}p'$. $i < 0$, hence $\mathsf{N}p'\#p \le 0$. □

Lemma 3.2.8 (Comparing two rational numbers)**.** $\forall^{\mathrm{nc}}_{a,b}(T_{\mathbf{Q}}a \to T_{\mathbf{Q}}b \to a \le b \vee b \le a)$.

Proof. Let a and b be given and assume $T_{\mathbf{Q}}a$ and $T_{\mathbf{Q}}b$. By Lemma 3.2.7 we can determine $a - b \le 0 \vee 0 \le a - b$. If the left disjunct holds, we prove $a \le b$, otherwise we prove $b \le a$. We use the fact that $T_{\mathbf{Q}}(a-b)$ holds provided that $T_{\mathbf{Q}}a$ and $T_{\mathbf{Q}}b$ hold. □

Lemma 3.2.9 (Standard split of rational numbers)**.** $\forall^{\mathrm{nc}}_{a}(Qa \to -1 \le a \le 0 \vee -\frac{1}{2} \le a \le \frac{1}{2} \vee 0 \le a \le 1)$.

Proof. Let a be given and assume Qa, namely, $a \in \mathbb{I}$ and $T_{\mathbf{Q}}a$. By Lemma 3.2.8 we determine $a \le b_0 \vee b_0 \le a$ for any $-\frac{1}{2} \le b_0 \le 0$. Let b_0 be $-\frac{1}{3}$ in this proof. We consider two cases from the disjunction. If $a \le -\frac{1}{3}$, we have $a \in \mathbb{I}_{\mathsf{L}}$. If $-\frac{1}{3} \le a$, we derive $a \le \frac{1}{3} \vee \frac{1}{3} \le a$ again by Lemma 3.2.8. We consider two cases. If $a \le \frac{1}{3}$, $a \in \mathbb{I}_{\mathsf{M}}$ holds. If $\frac{1}{3} \le a$, $a \in \mathbb{I}_{\mathsf{R}}$ holds. □

From a finite segment of an SDS, we can compute a rational number which approximates the SDS. Such a program is extracted from a proof of $\forall^{\mathrm{nc}}_{a \in \mathbb{I}}({}^{\mathrm{co}}Ia \to \forall^{\mathrm{c}}_{n}\exists^{\mathrm{l}}_{b \in \mathbb{I}}(|a-b| \le 2^{-n}))$. The result is essentially the same as Proposition 3.3.9.

$$\lambda_a({}^{\mathrm{co}}\mathcal{R}\, a\, \lambda_b(\mathbf{if}\ (b \le -\tfrac{1}{3})\ \mathbf{then}\ \mathbf{in}_2^4(\mathsf{InR}\ (2b+1))\ \mathbf{else\ if}\ (b \le \tfrac{1}{3})\ \mathbf{then}\ \mathbf{in}_3^4(\mathsf{InR}\ 2b)\ \mathbf{else}\ \mathbf{in}_4^4(\mathsf{InR}\ (2b-1))))$$

Figure 3.1: $\mathtt{ratsds}^{\mathbf{Q}\to\mathbf{I}}$

3.2.3 Program Extraction

We extract from the proof of Proposition 3.2.6 the program `ratsds` of type $\mathbf{Q} \to \mathbf{I}$ shown in Figure 3.2.3. Assume that a rational number a in $[-1, 1]$ is given. The corecursion operator applies the costep term to a, then it determines which of $a \le -\frac{1}{3}$, $-\frac{1}{3} < a < \frac{1}{3}$ or $\frac{1}{3} \le a$ is the case. The result is a term of sum type $\mathbf{U} + (\mathbf{I} + \mathbf{Q}) + (\mathbf{I} + \mathbf{Q}) + (\mathbf{I} + \mathbf{Q})$. In our case, it is an injection of a value whether in the second, in the third or in the fourth position of the sum. Depending on where the value is, the corecursion operator chooses which constructor, i.e., $\mathsf{C_L}$, $\mathsf{C_M}$ or $\mathsf{C_R}$, comes next. Since the value is always the right injection of a rational number, the unfolding can carry on with this rational number.

3.2.4 Experiments

A rational number -1 is translated into an SDS as follows. Let M be the costep term, then $\mathtt{ratsds}(-1)$ unfolds as follows.

$$\begin{aligned}\mathtt{ratsds}(-1) &\mapsto {}^{\mathrm{co}}\mathcal{R}\,(-1)\,M\\ &\mapsto \mathsf{C_L}({}^{\mathrm{co}}\mathcal{R}\,(-1)\,M)\\ &\mapsto \mathsf{C_L}(\mathsf{C_L}({}^{\mathrm{co}}\mathcal{R}\,(-1)\,M))\\ &\mapsto \ldots\end{aligned}$$

For another rational number $\frac{11}{7}$, `ratsds` computes the following.

$$\mathtt{ratsds}(11\#7) \mapsto^* \mathsf{C_R}(\mathsf{C_R}(\mathsf{C_M}(\mathsf{C_M}(\mathsf{C_R}(\mathsf{C_M}(\mathsf{C_M}(\mathsf{C_R}(\mathsf{C_M}(\mathsf{C_M}({}^{\mathrm{co}}\mathcal{R}\,(4\#7)\,M))))))))))$$

3.3 Representations of Real Numbers

In this section we deal with representations of real numbers of type-1 and type-0. The type-1 representation of real numbers is Cauchy reals which we saw in Section 3.1.1. We extract from formal proofs programs to transform a real number of type-1 into a real number of type-0, and vice versa. In contrast to Section 3.2, we are going to deal with real numbers including irrational numbers. From the technical viewpoint, we make use of abstract theory due to Berger for reasoning about real numbers, whereas our case study in Section 3.2 was based on the concrete rational numbers.

The rational numbers we used in Section 3.2 are concrete in the sense that they are ideals of the algebra **Q**. Notice that the rational numbers which appear in the extracted program `ratsds` are independent from the rational numbers which are used in predicate definitions and proofs. According to our realizability interpretation, arguments of predicates are irrelevant to realizers. Hence whatever the terms in formulas and proofs are, the extracted term is not affected by them at all. In fact, the algebra **Q** which appears in the extracted program `ratsds` comes from the associated algebra $\tau(T_{\mathbf{Q}})$ in the definition of Q via the realizability interpretation, but not from rational numbers in proofs. Based on this observation, the rational numbers in proofs can be something else as long as we can work consistently.

In this section, we work with the axiomatic theory of real numbers instead of a concrete one, e.g. some theory of Cauchy reals. Here we make use of the idea of abstract theories in order to formulate the axiomatic theory of real numbers. When we specify a new axiom, the realizability interpretation of the axiom should be given. If the axiom comes from an inductive or a coinductive definition, the realizer is given as a special constant. Even if this is not the case, non-computational (or Harrop) formulas are harmless as axioms because the empty term is the trivial realizer of such formulas. We refer to theories given by non-computational axiom formulas as abstract theories. We define abstract theories by using abstract types. Abstract types are just used to identify the kind of objects we are talking about, thus type variables are suitable. The use of abstract theories results in a clear separation between computational and non-computational ingredients of objects, hence it is beneficial to the implementation work as well.

3.3.1 Definitions

We first define constants and axioms of our abstract theory of real number, then define a coinductive predicate. The idea is to have a theory to reason about real numbers in the standard interval $[-1, 1]$. We give the real number constant zero and some restricted arithmetic operation on real numbers. Then, a relation "$\in$" is introduced in order to mean that a real number is in a rational interval.

Let ρ be an abstract type of real numbers. The algebras **N**, **SD**, **Q** and etc. are all defined in Section 2.1. The totality predicate of **SD** is needed.

Definition 3.3.1 (The totality predicate of **SD**)**.** Define $T_{\mathbf{SD}}$, the totality predicate of **SD**, to be $\mu_X(X\,\mathsf{L}, X\,\mathsf{M}, X\,\mathsf{R})$. Its introduction and elimination axiom are as follows.

$$T_{\mathbf{SD}}\,\mathsf{L} \qquad (T_{\mathbf{SD}})_0^+ \qquad\qquad T_{\mathbf{SD}}\,\mathsf{M} \qquad (T_{\mathbf{SD}})_1^+$$
$$T_{\mathbf{SD}}\,\mathsf{R} \qquad (T_{\mathbf{SD}})_2^+$$

$$\forall_d^{\mathrm{nc}}(T_{\mathbf{SD}}d \to P\,\mathsf{L} \to P\,\mathsf{M} \to P\,\mathsf{R} \to Pd) \qquad (T_{\mathbf{SD}})^-$$

Suppose that a ranges over **Q**, n ranges over **N**, d ranges over **SD** and x, y range over abstract real numbers of type ρ.

Definition 3.3.2 (Z)**.** Z is a constant of type ρ.

We have the following arithmetic operations on ρ.

Definition 3.3.3 ($\frac{x+d}{2}$, $2x-d$)**.** Let x and d be of type ρ and **SD**, respectively. $\frac{x+d}{2}$ and $2x-d$ are terms of type ρ.

Definition 3.3.4 ($x \in \mathbb{I}_{a,n}$)**.** Let x, a and n be of type ρ, **Q** and **N**, respectively, then $x \in \mathbb{I}_{a,n}$ is a term of type **B**. We abbreviate $x \in \mathbb{I}_{0,0}$ and $x \in \mathbb{I}_{\frac{d}{2},1}$ by $x \in \mathbb{I}$ and $x \in \mathbb{I}_d$, respectively.

Intuitively, $x \in \mathbb{I}_{a,n}$ means $a - 2^{-n} \leq x \leq a + 2^{-n}$. We refer to $\mathbb{I}_d$ by *basic intervals*.

Suppose that $\leq$ is the "equal or less than" relation on **Q** and eqd is the Leibniz equality. In formulas, we identify terms t of type **B** and the Leibniz equality $t \operatorname{eqd} \mathsf{T}$. The axioms of the abstract theory of real numbers are given as follows. Recall that the signed digits are taken to be integers, e.g. M to be 0.

Axiom 3.3.5 (Abstract Real Numbers)**.**

$$\forall_x(x \in \mathbb{I}), \qquad \text{(RealBound)}$$
$$\tfrac{\mathsf{Z}+\mathsf{M}}{2} \operatorname{eqd} \mathsf{Z}. \qquad \text{(AvZero)}$$
$$\forall_{x,a}(a \leq -\tfrac{1}{4} \to x \in \mathbb{I}_{a,2} \to x \in \mathbb{I}_{\mathsf{L}}), \qquad \text{(RealLeft)}$$
$$\forall_{x,a}(-\tfrac{1}{4} \leq a \leq \tfrac{1}{4} \to x \in \mathbb{I}_{a,2} \to x \in \mathbb{I}_{\mathsf{M}}), \qquad \text{(RealMiddle)}$$
$$\forall_{x,a}(\tfrac{1}{4} \leq a \to x \in \mathbb{I}_{a,2} \to x \in \mathbb{I}_{\mathsf{R}}), \qquad \text{(RealRight)}$$
$$\forall_{x,d}(x \in \mathbb{I}_d \to x \operatorname{eqd} \tfrac{(2x-d)+d}{2}), \qquad \text{(AvVaIdent)}$$
$$\forall_{x,d,a,n}(x \in \mathbb{I}_d \to x \in \mathbb{I}_{a,n+1} \to 2x - d \in \mathbb{I}_{2a-d,n}), \qquad \text{(VaIntro)}$$
$$\forall_{x,d,a,n}(x \in \mathbb{I}_{a,n} \to \tfrac{x+d}{2} \in \mathbb{I}_{\frac{a+d}{2},n+1}). \qquad \text{(AvIntro)}$$

Notice that all formulas of Axiom 3.3.5 are non-computational.

We consider real numbers in the interval $[-1, 1]$. This is characterized by (RealBound). The property of the constant Z is given by (AvZero). Informally, it means $\frac{0+0}{2} = 0$ as we commonly expect. The properties of the relation $x \in \mathbb{I}_{a,n}$ come from the rest of the axioms. The three axioms, (RealLeft), (RealMiddle) and (RealRight), characterize the relation between a real number and an interval. In each case, a basic interval is decided from the center a of the rational interval $\mathbb{I}_{a,n}$. Since a is a rational number, the inequalities in the first premises of the axioms are decidable. The last three axioms, (AvVaIdent), (VaIntro) and (AvIntro), characterize the properties of the arithmetic operations which also interact with rational intervals.

There is no computational content in abstract real numbers, but there is in the coinductively defined predicate ${}^{\mathrm{co}}I$. Due to the realizability interpretation, the argument of the predicate ${}^{\mathrm{co}}I$ is discarded by program extraction. The associated algebra of ${}^{\mathrm{co}}I$ is **I**, a cototal ideal of which represents a real number in $[-1, 1]$ through an SDS. The following definition differs from Definition 3.2.3 in the use of abstract real numbers.

Definition 3.3.6 ($^{\mathrm{co}}I$)**.** Define a coinductive predicate $^{\mathrm{co}}I$ of arity (ρ) to be

$$\nu_X(X\mathrm{Z}, \forall_x^{\mathrm{nc}}\forall_d^{\mathrm{c}}(Xx \to^{\mathrm{c}} X(\tfrac{x+d}{2}))).$$

The closure and greatest-fixed-point axioms are as follows.

$$\forall_x^{\mathrm{nc}}({}^{\mathrm{co}}Ix \to x \operatorname{eqd} \mathrm{Z} \vee^{\mathrm{r}} \exists_y^{\mathrm{r}}\exists_d^{\mathrm{d}}({}^{\mathrm{co}}Iy \wedge^{\mathrm{l}} x \operatorname{eqd} \tfrac{y+d}{2})), \qquad {}^{\mathrm{co}}I^-$$

$$\forall_x^{\mathrm{nc}}(Px \to \forall_x^{\mathrm{nc}}(Px \to x \operatorname{eqd} \mathrm{Z} \vee^{\mathrm{r}} \exists_y^{\mathrm{r}}\exists_d^{\mathrm{d}}(({}^{\mathrm{co}}Iy \vee^{\mathrm{d}} Py) \wedge^{\mathrm{l}} x \operatorname{eqd} \tfrac{y+d}{2})) \to {}^{\mathrm{co}}Ix). \qquad {}^{\mathrm{co}}I^+$$

This coinductively defined predicate $^{\mathrm{co}}I$ tells us the approximability of abstract real numbers via intervals.

3.3.2 Proofs

We prove two main propositions. From the proofs of the propositions, we will extract translators from a fast Cauchy real into an SDS and vice versa. Here, a fast Cauchy real is a Cauchy real whose Cauchy modulus is the identity function.

Let an abstract real number x be given. The premise of the first proposition claims that for any given precision n, there is a rational number a which approximates the x. The proof depends on Lemma 3.3.8 which we prove later.

Proposition 3.3.7 (Cauchy to SDS)**.** $\forall_x^{\mathrm{nc}}(\forall_n^{\mathrm{c}}\exists_a^{\mathrm{l}}(x \in \mathbb{I}_{a,n}) \to {}^{\mathrm{co}}Ix)$.

Proof. Let x be given and assume $\forall_n^{\mathrm{c}}\exists_a^{\mathrm{l}}(x \in \mathbb{I}_{a,n})$. Define the competitor predicate $Q = \{x \mid \forall_n^{\mathrm{c}}\exists_a^{\mathrm{l}}(x \in \mathbb{I}_{a,n})\}$ and apply $^{\mathrm{co}}I^+$. It suffices to show the costep formula $\forall_x^{\mathrm{nc}}(Qx \to x \operatorname{eqd} \mathrm{Z} \vee^{\mathrm{r}} \exists_y^{\mathrm{r}}\exists_d^{\mathrm{d}}(({}^{\mathrm{co}}Iy \vee^{\mathrm{d}} Qy) \wedge^{\mathrm{l}} x \operatorname{eqd} \tfrac{y+d}{2}))$. Let x be given and assume Qx, i.e., $\forall_n^{\mathrm{c}}\exists_a^{\mathrm{l}}(x \in \mathbb{I}_{a,n})$, hence $\exists_a^{\mathrm{l}}(x \in \mathbb{I}_{a,2})$ which then yields some d' such that $x \in \mathbb{I}_{d'}$ by Lemma 3.3.8. We prove the right disjunct of the goal. Apply $(\exists^{\mathrm{r}})^+$ and $(\exists^{\mathrm{d}})^+$ with $2x - d'$ and d', respectively. For the left conjunct, we prove the right disjunct $Q(2x - d')$, i.e., $\forall_n^{\mathrm{c}}\exists_a^{\mathrm{l}}(2x - d' \in \mathbb{I}_{a,n})$. Let n_0 be given, then Qx implies that there is some a_0 such that $x \in \mathbb{I}_{a_0,n_0+1}$. By (VaIntro), $2x - d' \in \mathbb{I}_{2a_0-d',n_0}$ holds, hence $\exists_a^{\mathrm{l}}(2x - d' \in \mathbb{I}_{a,n_0})$ as we desired. For the right conjunct use (AvVaIdent). □

Lemma 3.3.8 claims the following. Suppose that a real number $x \in [-1, 1]$ is covered by a rational interval $[a - \frac{1}{4}, a + \frac{1}{4}]$. Then, we can determine a basic interval which covers x.

Lemma 3.3.8. $\forall_x^{\mathrm{nc}}(\exists_a^{\mathrm{l}}(x \in \mathbb{I}_{a,2}) \to \exists_d^{\mathrm{l}}(x \in \mathbb{I}_d))$.

Proof. Let x be given and assume $\exists_a^{\mathrm{l}}(x \in \mathbb{I}_{a,2})$, which yields some rational number a such that $x \in \mathbb{I}_{a,2}$ holds. Because it is decidable which of $a \le -\frac{1}{4}$, $-\frac{1}{4} \le a \le \frac{1}{4}$ or $\frac{1}{4} \le a$ holds by Lemma 3.2.8, we find a d for the conclusion. For example if $a \le -\frac{1}{4}$, $x \in \mathbb{I}_{\mathsf{L}}$ holds by (RealLeft). Recall that $\exists_a^{\mathrm{l}}A$ abbreviates $\exists_a^{\mathrm{r}}(T_{\mathbf{Q}}a \wedge^{\mathrm{l}} A)$. □

From a proof of the converse of Proposition 3.3.7, we can extract a program which translates an SDS into a Cauchy real.

Proposition 3.3.9 (SDS to Cauchy)**.** $\forall_x^{\mathrm{nc}}({}^{\mathrm{co}}Ix \to \forall_n^{\mathrm{c}}\exists_a^{\mathrm{l}}(x \in \mathbb{I}_{a,n}))$.

Proof. The proposition is same as $\forall_n^{\mathrm{c}}\forall_x^{\mathrm{nc}}({}^{\mathrm{co}}Ix \to \exists_a^{\mathrm{l}}(x \in \mathbb{I}_{a,n}))$. We prove by induction on n. *Case* $n = 0$. Let x be given and assume ${}^{\mathrm{co}}Ix$. To prove $\exists_a^{\mathrm{l}}(x \in \mathbb{I}_{a,0})$, let a be 0. Then $x \in \mathbb{I}_{0,0}$ by (RealBound). *Case* $n \mapsto n+1$. Assume the induction hypothesis $\forall_x^{\mathrm{nc}}({}^{\mathrm{co}}Ix \to \exists_a^{\mathrm{l}}(x \in \mathbb{I}_{a,n}))$. We prove $\forall_x^{\mathrm{nc}}({}^{\mathrm{co}}Ix \to \exists_a^{\mathrm{l}}(x \in \mathbb{I}_{a,n+1}))$. Let x be given and assume ${}^{\mathrm{co}}Ix$, then ${}^{\mathrm{co}}I^-$ implies $x \operatorname{eqd} \mathsf{Z} \vee^{\mathrm{r}} \exists_y^{\mathrm{r}}\exists_d^{\mathrm{d}}({}^{\mathrm{co}}Iy \wedge^{\mathrm{l}} x \operatorname{eqd} \frac{y+d}{2})$. For the left case, suppose that $x \operatorname{eqd} \mathsf{Z}$. By the induction hypothesis there is a_0 such that $\mathsf{Z} \in \mathbb{I}_{a_0,n}$. Using (AvIntro) with M, $\frac{\mathsf{Z}+\mathsf{M}}{2} \in \mathbb{I}_{\frac{a_0+\mathsf{M}}{2},n+1}$ holds. Then the conclusion is trivial by (AvZero). For the right case, there are y and d such that $x \operatorname{eqd} \frac{y+d}{2}$ and ${}^{\mathrm{co}}Iy$. By the induction hypothesis, there is an a' such that $y \in \mathbb{I}_{a',n}$. By (AvIntro), $\frac{y+d}{2} \in \mathbb{I}_{\frac{a'+d}{2},n+1}$, hence the goal, $\exists_a^{\mathrm{l}}(x \in \mathbb{I}_{a,n+1})$. □

3.3.3 Program Extraction

We extract from the proof of Proposition 3.3.7 a program `cauchysds` which translates a fast Cauchy real into a signed digit stream. Corresponding to the use of Lemma 3.3.8 in the proof of the proposition, there is another program `L`, which is extracted from the lemma, appearing inside of `cauchysds`. By convention, we denote a fast Cauchy real of type $\mathbf{N} \to \mathbf{Q}$ by x. Note that variables of abstract type ρ are discarded by program extraction, hence don't appear in extracted terms at all. The extracted programs `cauchysds` and `L` are shown in Figure 3.2. In `cauchysds`, the given fast Cauchy real x is passed to the corecursion

$$
\begin{aligned}
&\texttt{cauchysds} = \lambda_{x^{\mathbf{N}\to\mathbf{Q}}}({}^{\mathrm{co}}\mathcal{R}_{\mathbf{I}}^{\mathbf{N}\to\mathbf{Q}}\, x\, \lambda_x(\mathsf{InR}\langle \mathtt{L}\, x, \mathsf{InR}(\lambda_n(2x(n+1) - \mathtt{L}\, x))\rangle))\\
&\mathtt{L} = \lambda_x(\text{Case } (x(2) + \tfrac{1}{4}) \text{ of}\\
&\qquad i_0 \# p \to \text{Case } i_0 \text{ of}\\
&\qquad\qquad \mathsf{P}\ p_0 \to \text{Case } (x(2) - \tfrac{1}{4}) \text{ of}\\
&\qquad\qquad\qquad i_1 \# p_0 \to \text{Case } i_1 \text{ of}\\
&\qquad\qquad\qquad\qquad \mathsf{P}\ p_1 \to \mathsf{R}\\
&\qquad\qquad\qquad\qquad \mathsf{0} \to \mathsf{M}\\
&\qquad\qquad\qquad\qquad \mathsf{N}\ p_1 \to \mathsf{M}\\
&\qquad\qquad \mathsf{0} \to \mathsf{L}\\
&\qquad\qquad \mathsf{N}\ p_0 \to \mathsf{L})
\end{aligned}
$$

Figure 3.2: $\mathtt{cauchysds}^{(\mathbf{N}\to\mathbf{Q})\to\mathbf{I}}$ and $\mathtt{L}^{(\mathbf{N}\to\mathbf{Q})\to\mathbf{SD}}$

operator. When the corecursion operator unfolds, this x is given to the costep term which computes a term of type $\mathbf{U} + \mathbf{SD} \times (\mathbf{I} + (\mathbf{N} \to \mathbf{Q}))$, where the actual values are injected in the positions of $\mathbf{SD}$ and $\mathbf{N} \to \mathbf{Q}$. As we will see, `L` determines from the given x a signed digit d such that $x \in \mathbb{I}_d$. By using this signed digit, the corecursion operator outputs this d

as the head of the SDS and carries on with the fast Cauchy real $\lambda_n(2x(n+1)-d)$ for the next step. This Cauchy real is intended to take a closer look at x.

In L, for the given fast Cauchy real x it determines if either $x \in \mathbb{I}_{\mathsf{L}}$, $x \in \mathbb{I}_{\mathsf{M}}$ or $x \in \mathbb{I}_{\mathsf{R}}$. The procedure of L is based on rational number reasoning. The given fast Cauchy real x is applied to 2, then it results in a rational number, say a. This a approximates x with an error of at most 2^{-2}. In other words, x is covered by a rational interval of length $\frac{1}{2}$ whose center is a. Whatever the center a is, we can find a basic interval which covers x. This is computed by testing $a \leq -\frac{1}{4}$ and $\frac{1}{4} \leq a$.

$$\begin{aligned}\mathtt{sdscauchy} = \lambda_{v^{\mathbf{I}},n^{\mathbf{N}}}(\mathcal{R}_{\mathbf{N}}^{\mathbf{I}\to\mathbf{Q}}\, n \\ &\lambda_u 0 \\ &\lambda_{m,f,u}(\text{Case } (\mathcal{D}_{\mathbf{I}} u) \text{ of } \mathsf{inL}\ \mathsf{U} \to 0 \\ &\qquad\qquad \mathsf{inR}\ y^{\mathbf{SD}\times\mathbf{I}} \to \tfrac{f(\pi_1 y)+\pi_0 y}{2}) \\ &v)\end{aligned}$$

Figure 3.3: sdscauchy

As shown in Figure 3.3, we extract from the proof of Proposition 3.3.9 a program sdscauchy which translates a signed digit stream into a fast Cauchy real. Informally speaking, this program computes from the given SDS $ds = d_0::d_1::\ldots::d_n::\ldots$ the following.

$$\lambda_n \sum_{i=0}^{n} \left(\frac{d_i}{2^{i+1}}\right)$$

Suppose that an SDS ds and a natural number n is given, then it computes the rational approximation of the given SDS by recursion on n. In one recursion step it destructs ds and observes nothing or a pair of d and ds', namely, the head and the tail of ds. In the former case it results in 0. In the latter case, we first assume that the previous value p of the recursion is available, then it computes $\frac{p+d}{2}$ where d is viewed as an integer.

3.3.4 Experiments

We define the square root of a rational number by recursion, then translate them into SDS by means of cauchysds. The following function sqrt computes a Cauchy real of the square root of a rational number with a modulus $M\, n := n+1$.

$$\begin{aligned}&\text{sqrt} : \mathbf{Q} \to \mathbf{N} \to \mathbf{Q} \\ &\text{sqrt} := \lambda_{a,n}\left(\mathcal{R}_{\mathbf{N}}\, n\, 1\, \lambda_{\_,b}\left(\frac{b+\frac{a}{b}}{2}\right)\right)\end{aligned}$$

The Cauchy reals representing $\sqrt{\frac{1}{2}}$ and $\sqrt{\frac{1}{3}}$ are defined to be $\mathrm{sqrt}\frac{1}{2}$ and $\mathrm{sqrt}\frac{1}{3}$, respectively. By means of `cauchysds`, they are translated into SDSs as follows:

$$\begin{aligned}\texttt{cauchysds}(\mathrm{sqrt}\tfrac{1}{2}) &= \mathsf{R} :: \mathsf{R} :: \mathsf{M} :: \mathsf{L} :: \mathsf{R} :: \mathsf{L} :: \mathsf{M} :: \mathsf{R} :: \mathsf{M} :: \mathsf{M} :: \ldots,\\ \texttt{cauchysds}(\mathrm{sqrt}\tfrac{1}{3}) &= \mathsf{R} :: \mathsf{M} :: \mathsf{R} :: \mathsf{L} :: \mathsf{M} :: \mathsf{R} :: \mathsf{M} :: \mathsf{M} :: \mathsf{M} :: \mathsf{L} :: \ldots.\end{aligned}$$

It has taken around a minute to compute the first 20 digits on the author's computer. In the opposite direction, `sdscauchy` computes a Cauchy real from an SDS. We give the first digits from the above results and 20 to `sdscauchy`.

$$\begin{aligned}&\texttt{sdscauchy}\,(\text{"the first digits of } \sqrt{\tfrac{1}{2}}\text{"})\,20 = 741455\#1048576\\ &\texttt{sdscauchy}\,(\text{"the first digits of } \sqrt{\tfrac{1}{3}}\text{"})\,20 = 151349\#262144\end{aligned}$$

As specified in Proposition 3.3.9, the results are rational approximations with the numerical error less than 2^{-20}.

$$\begin{aligned}\left|\tfrac{741455}{1048576} - \sqrt{\tfrac{1}{2}}\right| &= 1.90915\ldots \times 10^{-7} < 2^{-20}\\ \left|\tfrac{151349}{262144} - \sqrt{\tfrac{1}{3}}\right| &= 3.47265\ldots \times 10^{-7} < 2^{-20}\end{aligned}$$

3.4 Average

We develop elementary arithmetic in exact real numbers via program extraction. Instead addition of two real numbers, we rather consider the average of two real numbers so that the operation is closed in the interval $[-1, 1]$. The program of average comes from a proof of the following proposition

$$\forall^{\mathrm{nc}}_{x,y}({}^{\mathrm{co}}Ix \to {}^{\mathrm{co}}Iy \to {}^{\mathrm{co}}I(\tfrac{x+y}{2})). \tag{3.2}$$

The extracted term, which is of type $\mathbf{I} \to \mathbf{I} \to \mathbf{I}$, computes an average of two SDSs. For the brevity, assume two input ds and es of type $\mathbf{I}$ are of the form $d_0 :: d_1 :: d_2 :: \ldots$ and $e_0 :: e_1 :: e_2 :: \ldots$, respectively. The average $\tilde{d}_0 :: \tilde{d}_1 :: \tilde{d}_2 :: \ldots$ is computed by the extracted program step by step from the left end, i.e., $\tilde{d}_0$, to the right digits. Here we introduce notations. Consider an SDS $ds = d_0 :: d_1 :: d_2 :: \ldots$. Then, ds_0 and ds' stand for the head d_0 and the tail $d_1 :: d_2 :: \ldots$, respectively. Also the n-th tail can be written as $ds^{(n)}$ instead of using "$'$" for n times. Firstly, it computes $i_0 := d_0 + e_0 \in [-2, 2]$. We call this i_0 the 0-th carry. Secondly, the corecursive step computes from i_0, ds' and es' the following four values: a digit $\tilde{d}_0$, the next carry i_1, ds'' and es''. The last three outputs are the inputs for the next corecursion step. In general, i_n, ds^{n+1} and es^{n+1} are given as inputs to the corecursive step.

We determine $\tilde{d}_n$ and i_{n+1} in the following way:

$$\tilde{d}_n = \begin{cases} -1 & \text{if } -6 \le d_{n+1} + e_{n+1} + 2i_n \le -2 \\ 0 & \text{if } -2 \le d_{n+1} + e_{n+1} + 2i_n \le 2 \\ 1 & \text{if } 2 \le d_{n+1} + e_{n+1} + 2i_n \le 6 \end{cases}, \tag{3.3}$$

$$i_{n+1} = d_{n+1} + e_{n+1} + 2i_n - 4\tilde{d}_n. \tag{3.4}$$

It is explained by considering intervals. To determine the first digit $\tilde{d}_0$, it is insufficient just to read off d_0 and e_0. A counter example is the inputs $1 :: \ldots$ and $0 :: \ldots$ which indicate two intervals $[0, 1]$ and $[-\frac{1}{2}, \frac{1}{2}]$. Their average is somewhere in an approximated interval $[-\frac{1}{4}, \frac{3}{4}]$. However, we cannot determine the first digit $\tilde{d}_0$ because neither of the basic intervals, i.e., $[-1, 0]$, $[-\frac{1}{2}, \frac{1}{2}]$, and $[0, 1]$, can approximate such a real number. It suffices to use five distinctive tokens to memorize this 1-length interval. The carry $i_0 := d_0 + e_0 \in [-2, 2]$ is responsible for it.

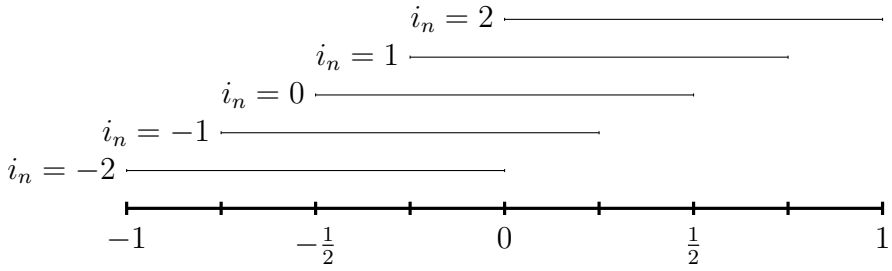

Figure 3.4: Intervals indicated by a carry $i_n \in \{-2, -1, 0, 1, 2\}$.

From now we take a look at our corecursive part. The average is approximated by $\frac{1}{2}$-length interval by reading off the digits once more from the two streams. This interval is represented by an integer $k_0 := d_1 + e_1 + 2i_0 \in [-6, 6]$. For example, assume $i_0 = -2$,

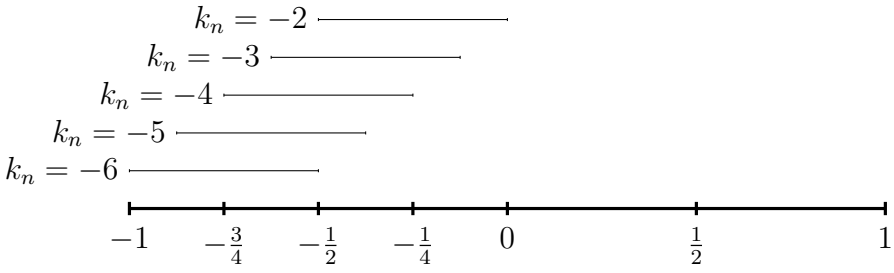

Figure 3.5: Intervals indicated by k_n for the case $i_n = -2$.

then the left most case happens if both of d_1 and e_1 are -1. There are five cases from

the combinations of d_1 and e_1. It is then possible to determine $\tilde{d}_0$ meaning either of $[-1,0]$, $[-\frac{1}{2},\frac{1}{2}]$ or $[0,1]$. We formalized this procedure to compute $\tilde{d}_n$ in (3.3). For further computation we reuse this corecursive part by mapping a $\frac{1}{2}$-length interval indicated by k_0 into a 1-length interval. It is done by applying $\mathrm{Out}_{\tilde{d}_0}(x) := 2x - \tilde{d}_0$. We find an i_1 which corresponds to the expected 1-length interval. For example in the case of $\tilde{d}_0 = -1$, $\mathrm{Out}_{-1}(x)$ maps each $\frac{1}{2}$-length interval indicated by $k_0 \in [-6,-2]$ into a 1-length interval indicated by $i_1 \in [-2,2]$, where $i_1 = k_0 - 4 \cdot (-1)$. We formalized this procedure to compute i_{n+1} in (3.4).

3.4.1 Definitions

According to our informal idea, the following extended signed digits suffice to be the carries.

Definition 3.4.1 (Extended signed digits)**.**

$$\mathbf{SD}_2 = \mu_\xi(\mathsf{LL}^\xi, \mathsf{LT}^\xi, \mathsf{MT}^\xi, \mathsf{RT}^\xi, \mathsf{RR}^\xi)$$

We define the addition on $\mathbf{SD}$ in $\mathbf{SD}_2$. Intuitively, it is clear by considering $\mathbf{SD}$ and $\mathbf{SD}_2$ in integers, $\{-1,0,1\}$ and $\{-2,-1,0,1,2\}$, respectively. Formally, we define the following function JOne.

Definition 3.4.2 (JOne)**.**

$$\begin{aligned}
&\mathrm{JOne} : \mathbf{SD} \to \mathbf{SD} \to \mathbf{SD}_2\\
&\mathrm{JOne}(\mathsf{L},\mathsf{L}) := \mathsf{LL}, \quad \mathrm{JOne}(\mathsf{L},\mathsf{M}) := \mathsf{LT}, \quad \mathrm{JOne}(\mathsf{L},\mathsf{R}) := \mathsf{MT},\\
&\mathrm{JOne}(\mathsf{M},\mathsf{L}) := \mathsf{LT}, \quad \mathrm{JOne}(\mathsf{M},\mathsf{M}) := \mathsf{MT}, \quad \mathrm{JOne}(\mathsf{M},\mathsf{R}) := \mathsf{RT},\\
&\mathrm{JOne}(\mathsf{R},\mathsf{L}) := \mathsf{MT}, \quad \mathrm{JOne}(\mathsf{R},\mathsf{M}) := \mathsf{RT}, \quad \mathrm{JOne}(\mathsf{R},\mathsf{R}) := \mathsf{RR}.
\end{aligned}$$

We abuse the common notation of addition $+$ for JOne.

We formalize the idea depicted in the introduction of this section. For the given d, e and i, we compute $d + e + 2i =: k \in [-6,6]$ which indicate an interval of length $\frac{1}{2}$, and we then determine d' and j for the next step. That idea yields the definitions of the following function J and D. In arithmetic formula, J' and D' can also be defined for the same purpose.

$$\begin{aligned}
&J' : \mathbf{Z} \to \mathbf{Z}, \qquad D' : \mathbf{Z} \to \mathbf{Z}\\
&J'(k) := \text{if } (|k| \bmod 4 = 3) \text{ then } (-\mathrm{sg}\ k) \text{ else } (\mathrm{sg}\ k) \cdot (|k| \bmod 4)\\
&D'(k) := \text{if } |k| \le 2 \text{ then } 0 \text{ else } (\mathrm{sg}\ k)\\
&\mathrm{sg}(k) := \begin{cases} -1 & k < 0 \\ 0 & k = 0 \\ 1 & k > 0 \end{cases}
\end{aligned}$$

We define functions J and D.

Definition 3.4.3 (J and D)**.** Define functions J and D of the following types,

$$J : \mathbf{SD} \to \mathbf{SD} \to \mathbf{SD}_2 \to \mathbf{SD}_2,$$
$$D : \mathbf{SD} \to \mathbf{SD} \to \mathbf{SD}_2 \to \mathbf{SD},$$

as shown in Figure 3.6 and Figure 3.7, respectively.

$J(\mathsf{L},\mathsf{L},\mathsf{LL}) := \mathsf{LL}$	$J(\mathsf{M},\mathsf{L},\mathsf{LL}) := \mathsf{LT}$	$J(\mathsf{R},\mathsf{L},\mathsf{LL}) := \mathsf{MT}$
$J(\mathsf{L},\mathsf{L},\mathsf{LT}) := \mathsf{MT}$	$J(\mathsf{M},\mathsf{L},\mathsf{LT}) := \mathsf{RT}$	$J(\mathsf{R},\mathsf{L},\mathsf{LT}) := \mathsf{LL}$
$J(\mathsf{L},\mathsf{L},\mathsf{MT}) := \mathsf{LL}$	$J(\mathsf{M},\mathsf{L},\mathsf{MT}) := \mathsf{LT}$	$J(\mathsf{R},\mathsf{L},\mathsf{MT}) := \mathsf{MT}$
$J(\mathsf{L},\mathsf{L},\mathsf{RT}) := \mathsf{MT}$	$J(\mathsf{M},\mathsf{L},\mathsf{RT}) := \mathsf{RT}$	$J(\mathsf{R},\mathsf{L},\mathsf{RT}) := \mathsf{RR}$
$J(\mathsf{L},\mathsf{L},\mathsf{RR}) := \mathsf{RR}$	$J(\mathsf{M},\mathsf{L},\mathsf{RR}) := \mathsf{LT}$	$J(\mathsf{R},\mathsf{L},\mathsf{RR}) := \mathsf{MT}$
$J(\mathsf{L},\mathsf{M},\mathsf{LL}) := \mathsf{LT}$	$J(\mathsf{M},\mathsf{M},\mathsf{LL}) := \mathsf{MT}$	$J(\mathsf{R},\mathsf{M},\mathsf{LL}) := \mathsf{RT}$
$J(\mathsf{L},\mathsf{M},\mathsf{LT}) := \mathsf{RT}$	$J(\mathsf{M},\mathsf{M},\mathsf{LT}) := \mathsf{LL}$	$J(\mathsf{R},\mathsf{M},\mathsf{LT}) := \mathsf{LT}$
$J(\mathsf{L},\mathsf{M},\mathsf{MT}) := \mathsf{LT}$	$J(\mathsf{M},\mathsf{M},\mathsf{MT}) := \mathsf{MT}$	$J(\mathsf{R},\mathsf{M},\mathsf{MT}) := \mathsf{RT}$
$J(\mathsf{L},\mathsf{M},\mathsf{RT}) := \mathsf{RT}$	$J(\mathsf{M},\mathsf{M},\mathsf{RT}) := \mathsf{RR}$	$J(\mathsf{R},\mathsf{M},\mathsf{RT}) := \mathsf{LT}$
$J(\mathsf{L},\mathsf{M},\mathsf{RR}) := \mathsf{LT}$	$J(\mathsf{M},\mathsf{M},\mathsf{RR}) := \mathsf{MT}$	$J(\mathsf{R},\mathsf{M},\mathsf{RR}) := \mathsf{RT}$
$J(\mathsf{L},\mathsf{R},\mathsf{LL}) := \mathsf{MT}$	$J(\mathsf{M},\mathsf{R},\mathsf{LL}) := \mathsf{RT}$	$J(\mathsf{R},\mathsf{R},\mathsf{LL}) := \mathsf{LL}$
$J(\mathsf{L},\mathsf{R},\mathsf{LT}) := \mathsf{LL}$	$J(\mathsf{M},\mathsf{R},\mathsf{LT}) := \mathsf{LT}$	$J(\mathsf{R},\mathsf{R},\mathsf{LT}) := \mathsf{MT}$
$J(\mathsf{L},\mathsf{R},\mathsf{MT}) := \mathsf{MT}$	$J(\mathsf{M},\mathsf{R},\mathsf{MT}) := \mathsf{RT}$	$J(\mathsf{R},\mathsf{R},\mathsf{MT}) := \mathsf{RR}$
$J(\mathsf{L},\mathsf{R},\mathsf{RT}) := \mathsf{RR}$	$J(\mathsf{M},\mathsf{R},\mathsf{RT}) := \mathsf{LT}$	$J(\mathsf{R},\mathsf{R},\mathsf{RT}) := \mathsf{MT}$
$J(\mathsf{L},\mathsf{R},\mathsf{RR}) := \mathsf{MT}$	$J(\mathsf{M},\mathsf{R},\mathsf{RR}) := \mathsf{RT}$	$J(\mathsf{R},\mathsf{R},\mathsf{RR}) := \mathsf{RR}$

Figure 3.6: Function J

Based on concrete rational numbers and integers, we define the abstract theory of average.

Axiom 3.4.4 (Abstract theory of average)**.** *Suppose that* $+_{\mathbf{Z}}$ *is the addition on* $\mathbf{Z}$*. We define the axioms of the abstract theory of average as follows.*

$$\begin{array}{ll}
\frac{x+k}{2} + l = \frac{x+(k+_{\mathbf{Z}}2l)}{2}, & x + 0 = x, \\
\frac{x+k}{4} + l = \frac{x+(k+_{\mathbf{Z}}4l)}{4}, & 0 + y = y, \\
\frac{x+k}{2} + \frac{y+l}{2} = \frac{(x+y)+(k+_{\mathbf{Z}}l)}{4}, & \frac{0}{2} = 0, \\
k + l = k +_{\mathbf{Z}} l, & \frac{2k}{2} = k.
\end{array}$$

In the proof, while we use equations in Axiom 3.4.4 in order to replace the left hand side by the right hand side, the opposite way is never needed in our proofs. Moreover,

$$
\begin{array}{lll}
D(\mathsf{L},\mathsf{L},\mathsf{LL}) := \mathsf{L} & D(\mathsf{M},\mathsf{L},\mathsf{LL}) := \mathsf{L} & D(\mathsf{R},\mathsf{L},\mathsf{LL}) := \mathsf{L} \\
D(\mathsf{L},\mathsf{L},\mathsf{LT}) := \mathsf{L} & D(\mathsf{M},\mathsf{L},\mathsf{LT}) := \mathsf{L} & D(\mathsf{R},\mathsf{L},\mathsf{LT}) := \mathsf{M} \\
D(\mathsf{L},\mathsf{L},\mathsf{MT}) := \mathsf{M} & D(\mathsf{M},\mathsf{L},\mathsf{MT}) := \mathsf{M} & D(\mathsf{R},\mathsf{L},\mathsf{MT}) := \mathsf{M} \\
D(\mathsf{L},\mathsf{L},\mathsf{RT}) := \mathsf{M} & D(\mathsf{M},\mathsf{L},\mathsf{RT}) := \mathsf{M} & D(\mathsf{R},\mathsf{L},\mathsf{RT}) := \mathsf{M} \\
D(\mathsf{L},\mathsf{L},\mathsf{RR}) := \mathsf{M} & D(\mathsf{M},\mathsf{L},\mathsf{RR}) := \mathsf{R} & D(\mathsf{R},\mathsf{L},\mathsf{RR}) := \mathsf{R} \\
D(\mathsf{L},\mathsf{M},\mathsf{LL}) := \mathsf{L} & D(\mathsf{M},\mathsf{M},\mathsf{LL}) := \mathsf{L} & D(\mathsf{R},\mathsf{M},\mathsf{LL}) := \mathsf{L} \\
D(\mathsf{L},\mathsf{M},\mathsf{LT}) := \mathsf{L} & D(\mathsf{M},\mathsf{M},\mathsf{LT}) := \mathsf{M} & D(\mathsf{R},\mathsf{M},\mathsf{LT}) := \mathsf{M} \\
D(\mathsf{L},\mathsf{M},\mathsf{MT}) := \mathsf{M} & D(\mathsf{M},\mathsf{M},\mathsf{MT}) := \mathsf{M} & D(\mathsf{R},\mathsf{M},\mathsf{MT}) := \mathsf{M} \\
D(\mathsf{L},\mathsf{M},\mathsf{RT}) := \mathsf{M} & D(\mathsf{M},\mathsf{M},\mathsf{RT}) := \mathsf{M} & D(\mathsf{R},\mathsf{M},\mathsf{RT}) := \mathsf{R} \\
D(\mathsf{L},\mathsf{M},\mathsf{RR}) := \mathsf{R} & D(\mathsf{M},\mathsf{M},\mathsf{RR}) := \mathsf{R} & D(\mathsf{R},\mathsf{M},\mathsf{RR}) := \mathsf{R} \\
D(\mathsf{L},\mathsf{R},\mathsf{LL}) := \mathsf{L} & D(\mathsf{M},\mathsf{R},\mathsf{LL}) := \mathsf{L} & D(\mathsf{R},\mathsf{R},\mathsf{LL}) := \mathsf{M} \\
D(\mathsf{L},\mathsf{R},\mathsf{LT}) := \mathsf{M} & D(\mathsf{M},\mathsf{R},\mathsf{LT}) := \mathsf{M} & D(\mathsf{R},\mathsf{R},\mathsf{LT}) := \mathsf{M} \\
D(\mathsf{L},\mathsf{R},\mathsf{MT}) := \mathsf{M} & D(\mathsf{M},\mathsf{R},\mathsf{MT}) := \mathsf{M} & D(\mathsf{R},\mathsf{R},\mathsf{MT}) := \mathsf{M} \\
D(\mathsf{L},\mathsf{R},\mathsf{RT}) := \mathsf{M} & D(\mathsf{M},\mathsf{R},\mathsf{RT}) := \mathsf{R} & D(\mathsf{R},\mathsf{R},\mathsf{RT}) := \mathsf{R} \\
D(\mathsf{L},\mathsf{R},\mathsf{RR}) := \mathsf{R} & D(\mathsf{M},\mathsf{R},\mathsf{RR}) := \mathsf{R} & D(\mathsf{R},\mathsf{R},\mathsf{RR}) := \mathsf{R}
\end{array}
$$

Figure 3.7: Function D

the right hand sides are simpler than the left hand sides. Due to these conditions, we can make use of the (proper) rewriting rules to specify the above axiom in Minlog. Then, the normalization mechanism automatically applies axioms instead of manually applying each axiom one by one, so that proof writing in Minlog becomes much easier.

3.4.2 Proofs

We present a proof of the formula (3.2) by closely following [BS10]. By convention on variable names, d and e are for $\mathbf{SD}$, and i and j are for $\mathbf{SD}_2$.

Proposition 3.4.5 (Average)**.** $\forall^{\mathrm{nc}}_{x,y}({}^{\mathrm{co}}Ix \to {}^{\mathrm{co}}Iy \to {}^{\mathrm{co}}I\frac{x+y}{2})$

Proof. Let x and y be given. Assume ${}^{\mathrm{co}}Ix$ and ${}^{\mathrm{co}}Iy$. By Lemma 3.4.6, there exist x', y' and i such that $x + y = \frac{x'+y'+i}{2}$. We prove ${}^{\mathrm{co}}I\frac{x'+y'+i}{4}$ by using $({}^{\mathrm{co}}I)^+$ with the competitor predicate $P := \{z|\exists^{\mathrm{r}}_{x,y}\exists^{\mathrm{d}}_i({}^{\mathrm{co}}Ix \wedge {}^{\mathrm{co}}Iy \wedge z = \frac{x+y+i}{4})\}$. Obviously $P\frac{x'+y'+i}{4}$ holds. It suffices to prove the following costep formula.

$$\forall^{\mathrm{nc}}_x(Px \to x = 0 \vee \exists^{\mathrm{r}}_y\exists^{\mathrm{d}}_d(({}^{\mathrm{co}}Iy \vee Py) \wedge x = \frac{y+d}{2})).$$

Let x be given. Assume Px. Here, it makes computational sense to use the identity lemma before assuming Px. By unfolding P, there exist $\tilde{x}$, $\tilde{y}$, i such that $x = \frac{\tilde{x}+\tilde{y}+i}{4}$, ${}^{\mathrm{co}}I\tilde{x}$,

${}^{\mathrm{co}}I\tilde{y}$, $T_{\mathbf{SD}_2}i$. By Lemma 3.4.7, there exist $\tilde{x}', \tilde{y}', j, d$ such that $\tilde{x} + \tilde{y} + i = \frac{\tilde{x}'+\tilde{y}'+j+4d}{2}$. Let $y = \frac{\tilde{x}'+\tilde{y}'+j}{4}$, then $x = \frac{y+d}{2}$ and Py hold. □

We prove the two lemmas used in the above proof. In the following we abbreviate $\forall^{\mathrm{nc}}_x({}^{\mathrm{co}}Ix \to A)$ and $\exists^{\mathrm{r}}_x({}^{\mathrm{co}}Ix \wedge A)$ by $\forall^{\mathrm{nc}}_{x\in{}^{\mathrm{co}}I}A$ and $\exists^{\mathrm{r}}_{x\in{}^{\mathrm{co}}I}(A)$, respectively.

Lemma 3.4.6. $\forall^{\mathrm{nc}}_{x,y\in{}^{\mathrm{co}}I}\exists^{\mathrm{d}}_i\exists^{\mathrm{r}}_{x',y'\in{}^{\mathrm{co}}I}\left(\frac{x+y}{2} = \frac{x'+y'+i}{4}\right)$.

Proof. Assume $x \in {}^{\mathrm{co}}I$ and $y \in {}^{\mathrm{co}}I$. Applying $({}^{\mathrm{co}}I)^-$ to ${}^{\mathrm{co}}Ix$ and ${}^{\mathrm{co}}Iy$, there are four cases from the following two disjunctions.

$$x \operatorname{eqd} \mathrm{Z} \vee^{\mathrm{r}} \exists^{\mathrm{r}}_{x_0\in{}^{\mathrm{co}}I}\exists^{\mathrm{d}}_{d_0}(x \operatorname{eqd} \frac{x_0 + d_0}{2}),$$
$$y \operatorname{eqd} \mathrm{Z} \vee^{\mathrm{r}} \exists^{\mathrm{r}}_{y_0\in{}^{\mathrm{co}}I}\exists^{\mathrm{d}}_{e_0}(y \operatorname{eqd} \frac{y_0 + e_0}{2}).$$

We prove the goal $\exists^{\mathrm{d}}_i\exists^{\mathrm{r}}_{x',y'\in{}^{\mathrm{co}}I} \frac{x+y}{2} \operatorname{eqd} \frac{x'+y'+i}{4}$ for each case. (1) Assume $x \operatorname{eqd} \mathrm{Z}$ and $y \operatorname{eqd} \mathrm{Z}$, then $\frac{x+y}{2}$ is Z. Let x', y' be Z and i be MT. (2) Assume $x \operatorname{eqd} \mathrm{Z}$ and $\exists^{\mathrm{r}}_{y_0\in{}^{\mathrm{co}}I}\exists^{\mathrm{d}}_{e_0}(y \operatorname{eqd} \frac{y_0+e_0}{2})$. For the given y_0 and e_0, $\frac{x+y}{2}$ is $\frac{y_0+e_0}{4}$. Let x' be Z, y' be y_0 and i be e_0. (3) Similar to (2). (4) Assume $x_0 \in {}^{\mathrm{co}}I$, $d_0 \in \mathbf{SD}$, $y_0 \in {}^{\mathrm{co}}I$, $e_0 \in \mathbf{SD}$ such that $x \operatorname{eqd} \frac{x_0+d_0}{2}$ and $y \operatorname{eqd} \frac{y_0+e_0}{2}$, then $\frac{x+y}{2}$ is $\frac{x_0+y_0+(d_0+e_0)}{4}$. Let x' be x_0, y' be y_0 and i be $d_0 + e_0$. □

Lemma 3.4.7. $\forall^{\mathrm{nc}}_{x,y\in{}^{\mathrm{co}}I}\forall^{\mathrm{c}}_i\exists^{\mathrm{r}}_{x',y'\in{}^{\mathrm{co}}I}\exists^{\mathrm{d}}_{j,d}(x + y + i = \frac{x'+y'+j+4d}{2})$.

Proof. Let x, y and i be given and assume ${}^{\mathrm{co}}Ix$ and ${}^{\mathrm{co}}Iy$. By using $({}^{\mathrm{co}}I)^-$, we have the same case distinction as in Lemma 3.4.6. (1) Suppose that $x \operatorname{eqd} \mathrm{Z}$ and $y \operatorname{eqd} \mathrm{Z}$. Let x' and y' be both Z, then it suffices to find j and d such that $i = \frac{j+4d}{2}$. It is solved by $j := J(i, 0, 0)$ and $d := D(i, 0, 0)$. (2) Suppose that $x \operatorname{eqd} \mathrm{Z}$ and $\exists^{\mathrm{r}}_{y_0\in{}^{\mathrm{co}}I}\exists^{\mathrm{d}}_{e_0}(y \operatorname{eqd} \frac{y_0+e_0}{2})$. Let x' be Z and y' be y_0, then it suffices to find j and d such that $\frac{e_0}{2} + i = \frac{j+4d}{2}$. The equation is solved by $j := J(i, 0, e_0)$ and $d := D(i, 0, e_0)$. (3) Similar to (2). (4) Suppose that $x_0 \in {}^{\mathrm{co}}I$, $d_0 \in \mathbf{SD}$, $y_0 \in {}^{\mathrm{co}}I$, $e_0 \in \mathbf{SD}$ such that $x \operatorname{eqd} \frac{x_0+d_0}{2}$ and $y \operatorname{eqd} \frac{y_0+e_0}{2}$. Let x' be x_0, y' be y_0, then it suffices to find j and d such that $\frac{d_0}{2} + \frac{e_0}{2} + i = \frac{j+4d}{2}$. The equation is solved by $j := J(i, d_0, e_0)$ and $d := D(i, d_0, e_0)$. □

3.4.3 Program Extraction

We extract programs from the proofs. From Lemma 3.4.6, we extract `cPreprocess` as shown in Figure 3.8. Here ds is a name for variables ranging over $\mathbf{I}$, and u for variables ranging over $\mathbf{SD} \times \mathbf{I}$. This `cPreprocess` satisfies the following equations.

$$\texttt{cPreprocess}([], []) = \langle \mathsf{M}, [], [] \rangle,$$
$$\texttt{cPreprocess}([], e{::}es) = \langle e, [], es \rangle,$$
$$\texttt{cPreprocess}(d{::}ds, []) = \langle d, ds, [] \rangle,$$
$$\texttt{cPreprocess}(d{::}ds, e{::}es) = \langle d + e, ds, es \rangle.$$

$$\begin{aligned}
&\lambda_{ds_0,ds_1}(\text{Case } \mathcal{D}_{\mathbf{I}} ds_0 \text{ of}\\
&\qquad \lambda_(\text{Case } \mathcal{D}_{\mathbf{I}} ds_1 \text{ of } \lambda_\langle \mathsf{MT}, ds_0, ds_1\rangle\\
&\qquad\qquad \lambda_u\langle \mathsf{M} + \pi_0 u, ds_0, \pi_1 u\rangle)\\
&\qquad \lambda_{u_0}(\text{Case } \mathcal{D}_{\mathbf{I}} ds_1 \text{ of } \lambda_\langle \pi_0 u_0 + \mathsf{M}, \pi_1 u_0, ds_1\rangle\\
&\qquad\qquad \lambda_{u_1}\langle \pi_0 u_0 + \pi_0 u_1, \pi_1 u_0, \pi_1 u_1\rangle))
\end{aligned}$$

Figure 3.8: $\texttt{cPreprocess}^{\mathbf{I}\to\mathbf{I}\to\mathbf{SD}_2\times\mathbf{I}\times\mathbf{I}}$

For the given two streams `cPreprocess` computes the sum of the two head digits (regarding [] as $\mathsf{M}::[]$), and its tails. This sum of digits of type $\mathbf{SD}_2$ is a carry which contains intermediate information to compute the average as we saw in the informal idea.

From the proof of Lemma 3.4.7, we extract the term `cRoutine` as shown in Figure 3.9. The constant `cRoutine` of type $\mathbf{SD}_2 \to \mathbf{I} \to \mathbf{I} \to \mathbf{SD}_2 \times \mathbf{SD} \times \mathbf{I} \times \mathbf{I}$ is defined to be the

$$\begin{aligned}
&\lambda_{i,ds_0,ds_1}(\text{Case } \mathcal{D}_{\mathbf{I}} ds_0 \text{ of}\\
&\qquad \lambda_(\text{Case } \mathcal{D}_{\mathbf{I}} ds_1 \text{ of}\\
&\qquad\qquad \lambda_\langle J\mathsf{MM}i, D\mathsf{MM}i, ds_0, ds_1\rangle\\
&\qquad\qquad \lambda_u\langle J\mathsf{M}(\pi_0 u)i, D\mathsf{M}(\pi_0 u)i, ds_0, \pi_1 u\rangle)\\
&\qquad \lambda_{u_0}(\text{Case } \mathcal{D}_{\mathbf{I}} ds_1 \text{ of}\\
&\qquad\qquad \lambda_\langle J\mathsf{M}(\pi_0 u_0), D(\pi_0 u_0)\mathsf{M}i, \pi_1 u_0, ds_1\rangle\\
&\qquad\qquad \lambda_{u_1}\langle J(\pi_0 u_0)(\pi_0 u_1)i, D(\pi_0 u)(\pi_0 u_1)i, \pi_1 u_0, \pi_1 u_1\rangle))
\end{aligned}$$

Figure 3.9: $\texttt{cRoutine}^{\mathbf{SD}_2\to\mathbf{I}\to\mathbf{I}\to\mathbf{SD}_2\times\mathbf{SD}\times\mathbf{I}\times\mathbf{I}}$

term above. It satisfies the equations

$$\begin{aligned}
&\texttt{cRoutine}(i, [], []) = \langle J(0,0,i), D(0,0,i), [], []\rangle,\\
&\texttt{cRoutine}(i, [], e::es) = \langle J(0,e,i), D(0,e,i), [], es\rangle,\\
&\texttt{cRoutine}(i, d::ds, []) = \langle J(d,0,i), D(d,0,i), ds, []\rangle,\\
&\texttt{cRoutine}(i, d::ds, e::es) = \langle J(d,e,i), D(d,e,i), ds, es\rangle.
\end{aligned}$$

For the given carry and two signed digit streams, `cRoutine` computes the carry for the next step, the first signed digit of the average of the streams, and the tails of the input streams.

From the proof of Proposition 3.4.5, we extract `ave` as shown in Figure 3.10. Here q is a name for variables ranging over $\mathbf{SD}_2\times\mathbf{I}\times\mathbf{I}$ and w for variables ranging over $\mathbf{SD}_2\times\mathbf{SD}\times\mathbf{I}\times\mathbf{I}$. Also recall that the let construction in Example 2.5.28. Note that we have three times of `cRoutine` if we do not use the identity lemma in the proof of Proposition 3.4.5. It calls

$$\lambda_{ds_0,ds_1}({}^{\mathrm{co}}\mathcal{R}\,(\mathtt{cPreprocess}\,ds_0\,ds_1)\\ \lambda_q(\mathsf{InL}(\mathbf{let}\,w\,=\,\mathtt{cRoutine}(\pi_0 q)(\pi_0(\pi_1\,q))(\pi_1(\pi_1 q))\\ \mathbf{in}\,\langle\pi_0(\pi_1 w),\mathsf{InR}\langle\pi_0 w,\pi_1(\pi_1 w)\rangle\rangle))))$$

Figure 3.10: $\mathtt{ave}^{\mathbf{I}\to\mathbf{I}\to\mathbf{I}}$

`cPreprocess` to compute the first carry and the tails of the inputs. Then `CoRec` repeatedly calls `cRoutine` in order to compute the average step by step. The second argument of the corecursion operator, namely, $M\colon \mathbf{SD}_2\times\mathbf{I}\times\mathbf{I}\to\mathbf{U}+\mathbf{SD}\times(\mathbf{I}+\mathbf{SD}_2\times\mathbf{I}\times\mathbf{I})$, operates on an argument $\langle i, ds, es\rangle$ as follows. Let $\mathtt{cRoutine}(i, ds, es) = \langle j, \tilde{d}, ds', es'\rangle$. Then $M\langle i, ds, es\rangle = \mathsf{InR}\langle\tilde{d}, \mathsf{InR}\langle j, ds', es'\rangle\rangle$. Given ds and es, let $\mathtt{cPreprocess}(ds, es) = \langle i, ds, es\rangle \coloneqq N$. Then MN is $\mathsf{InR}\langle\tilde{d}, \mathsf{InR}\langle j, ds', es'\rangle\rangle$. Therefore by the conversion rule for the corecursion operator the result is $\tilde{d}::({}^{\mathrm{co}}\mathcal{R}\langle j, ds', es'\rangle M)$.

The term above can be used as a program to compute the average of stream represented real numbers. Comparing with a result by Berger and Seisenberger [BS12], our extracted program behaves almost the same as theirs except for a difference coming from the data type of streams. They use the type of necessarily infinite streams, written as $\mathrm{fix}\,\alpha.\mathrm{SD}\times\alpha$, whereas we accept possibly infinite streams.

3.4.4 Experiments

We compute the average of $\frac{5}{8}$ and $\frac{3}{4}$. $\frac{5}{8} = \frac{1}{2}+\frac{0}{4}+\frac{1}{8}$ and $\frac{3}{4} = \frac{1}{2}+\frac{1}{4}$, and hence those numbers are $\mathsf{R}::\mathsf{M}::\mathsf{R}::[]$ and $\mathsf{R}::\mathsf{R}::[]$ in the algebra $\mathbf{I}$, respectively. Unfolding up to 10 times, the average program results in

$$\mathsf{R}::\mathsf{R}::\mathsf{M}::\mathsf{L}::\mathsf{M}::\mathsf{M}::\mathsf{M}::\mathsf{M}::\mathsf{M}::\mathsf{M}::({}^{\mathrm{co}}\mathcal{R}\langle\mathsf{M}, [], []\rangle M).$$

It is verified as

$$\frac{\frac{5}{8}+\frac{3}{4}}{2} = \frac{11}{16} = \frac{1}{2}+\frac{1}{4}-\frac{1}{16}.$$

Next, we try computing the average of two irrational numbers, $\frac{1}{\sqrt{2}}$ and $\frac{1}{\sqrt{3}}$. By means of `cauchysds`, which we extracted in Section 3.3, the first digits of these numbers are the following.

$$\mathtt{cauchysds}(\mathrm{sqrt}\tfrac{1}{2}) = \mathsf{R}::\mathsf{R}::\mathsf{M}::\mathsf{L}::\mathsf{R}::\mathsf{L}::\mathsf{M}::\mathsf{R}::\mathsf{M}::\mathsf{M}::\ldots,$$
$$\mathtt{cauchysds}(\mathrm{sqrt}\tfrac{1}{3}) = \mathsf{R}::\mathsf{M}::\mathsf{R}::\mathsf{L}::\mathsf{M}::\mathsf{R}::\mathsf{M}::\mathsf{M}::\mathsf{M}::\mathsf{L}::\ldots.$$

The average of them are computed to be

$$\mathsf{R}::\mathsf{R}::\mathsf{L}::\mathsf{M}::\mathsf{M}::\mathsf{R}::\mathsf{M}::\mathsf{M}::\mathsf{R}::\mathsf{M}::\ldots.$$

To verify the error, we first transform it into rational numbers, 329#512. The error is

$$\left|\frac{\frac{1}{\sqrt{2}}+\frac{1}{\sqrt{3}}}{2}-\frac{329}{512}\right| = 3.49\ldots\times 10^{-4},$$

which is smaller than $2^{-10} = 9.765625 \times 10^{-4}$ as expected.

3.5 Representations of Uniformly Continuous Functions

We already saw a constructive representation of uniformly continuous functions in Section 3.1. There, the representation of uniformly continuous functions involves first order function types, hence it is called the type-1 representation of uniformly continuous functions. In this section, we study a representation of unary uniformly continuous functions by means of coinduction. We refer to this by type-0 uniformly continuous functions, since they are ground type objects in contrast to type-1 uniformly continuous functions. Our main result in this section is an extracted program which translates a type-1 uniformly continuous function into a type-0 uniformly continuous function, and vice versa.

Recall that $\langle h^{\mathbf{Q}\to\mathbf{N}\to\mathbf{Q}}, \alpha^{\mathbf{N}\to\mathbf{N}}, \omega^{\mathbf{N}\to\mathbf{N}}\rangle$ defines a type-1 uniformly continuous function if the following conditions are satisfied.

$$\forall_{f,a^{\mathbf{Q}},k^{\mathbf{N}},n\geq\alpha(k),m\geq\alpha(k)}(|h(a,n)-h(a,m)|\leq 2^{-k}),$$
$$\forall_{f,a^{\mathbf{Q}},b^{\mathbf{Q}},k^{\mathbf{N}},n\geq\alpha(k)}(|a-b|\leq 2^{-\omega(k)+1}\to|h(a,n)-h(b,n)|\leq 2^{-k}).$$

As our type-0 uniformly continuous functions, we adopt so-called read-write machines [Ber11] or stream processors [GHP06, HPG09]. It is a $\mathbf{W}$-cototal $\mathbf{R}_{\mathbf{W}}$-total ideal, where algebras $\mathbf{R}_\alpha$ and $\mathbf{W}$ are defined by

$$\mathbf{R}_\alpha := \mu_\xi(\mathsf{Put}^{\mathbf{SD}\to\alpha\to\xi}, \mathsf{Get}^{\xi\to\xi\to\xi\to\xi}),$$
$$\mathbf{W} := \mu_\xi(\mathsf{Stop}^{\xi}, \mathsf{Cont}^{\mathbf{R}_\xi\to\xi}).$$

A $\mathbf{W}$-cototal $\mathbf{R}_{\mathbf{W}}$-total ideal is a non-well founded tree which consists of total trees of type $\mathbf{R}_{\mathbf{W}}$. It represents a translation of an SDS into another in the following way. The root of the tree is either Stop or $\mathsf{Cont}\,r$. If it is Stop, the rest of the input stream goes to the output stream. If it is $\mathsf{Cont}\,r$, we continue with r which is either $\mathsf{Put}\,d\,w$ or $\mathsf{Get}\,r_{\mathsf{L}}\,r_{\mathsf{M}}\,r_{\mathsf{R}}$. If it is $\mathsf{Put}\,d\,w$, it outputs one signed digit d and continue with w. If it is $\mathsf{Get}\,r_{\mathsf{L}}\,r_{\mathsf{M}}\,r_{\mathsf{R}}$, it reads one signed digit $e^{\mathbf{SD}}$ from the input and continue with r_e. Observe that there is a nested structure in the procedure: the outer repetition on $\mathbf{W}$ and the inner one on $\mathbf{R}_{\mathbf{W}}$. It is crucial to have a well founded structure inside of a non-well founded structure in order to ensure the uniform continuity. We define a read-write machine to be a $\mathbf{W}$-cototal $\mathbf{R}_{\mathbf{W}}$-total ideal which can produce infinite signed digits, whereas its each internal component, i.e., an $\mathbf{R}_{\mathbf{W}}$-total ideal, has to put one signed digit by Put after getting finite signed digits by Get.

3.5.1 Definitions

We describe the abstract theory of uniformly continuous functions. Let ϕ be an abstract type of uniformly continuous functions. We define constants using this abstract type ϕ, then the properties are specified via non-computational axioms.

Suppose that p, q range over $\mathbf{Q}$, k, l range over $\mathbf{N}$, d ranges over $\mathbf{SD}$ and f, g range over ϕ. We define constants for the identity function and arithmetic operations Out and In on functions such that intuitively $(\mathrm{Out}_d \circ f)(x) = 2f(x) - d$ and $(f \circ \mathrm{In}_d)(x) = f(\frac{x+d}{2})$ hold.

Definition 3.5.1 (Id)**.** Id is a constant of type ϕ.

Definition 3.5.2 (Out, In)**.** Out and In are constants of type $\mathbf{SD} \to \phi \to \phi$. Out $d\, f$ and In $d\, f$ are abbreviated as $\mathrm{Out}_d \circ f$ and $f \circ \mathrm{In}_d$, respectively.

We also define an abstract inclusion relation on intervals.

Definition 3.5.3 (Sub)**.** Sub is a constant of type $\phi \to \mathbf{Q} \to \mathbf{N} \to \mathbf{Q} \to \mathbf{N} \to \mathbf{B}$. Sub $f\ p\ l\ q\ k$ is abbreviated as $f[\mathbb{I}_{p,l}] \subseteq \mathbb{I}_{q,k}$.

The axioms of the abstract theory of uniformly continuous functions are defined as follows.

Axiom 3.5.4 (Abstract Theory of Uniformly Continuous Functions)**.**

$$\forall_{p,l}(\mathrm{Id}[\mathbb{I}_{p,l}] \subseteq \mathbb{I}_{p,l}), \tag{UcfId}$$
$$\forall_{f,p}(f[\mathbb{I}_{p,0}] \subseteq \mathbb{I}), \tag{UcfBound}$$
$$\forall_{f,p,l,q,k}(f[\mathbb{I}_{p,l}] \subseteq \mathbb{I}_{q,k} \to f[\mathbb{I}_{p,l+1}] \subseteq \mathbb{I}_{q,k}), \tag{UcfInputSucc}$$
$$\forall_{f,d,p,l,q,k}(f[\mathbb{I}] \subseteq \mathbb{I}_d \to (\mathrm{Out}_d \circ f)[\mathbb{I}_{p,l}] \subseteq \mathbb{I}_{q,k} \to f[\mathbb{I}_{p,l}] \subseteq \mathbb{I}_{\frac{q+d}{2},k+1}), \tag{OutElim}$$
$$\forall_{f,d,p,l,q,k}(f[\mathbb{I}] \subseteq \mathbb{I}_d \to f[\mathbb{I}_{p,l}] \subseteq \mathbb{I}_{\frac{q+d}{2},k+1} \to (\mathrm{Out}_d \circ f)[\mathbb{I}_{p,l}] \subseteq \mathbb{I}_{q,k}), \tag{OutIntro}$$
$$\forall_{f,d,p,l,q,k}((f \circ \mathrm{In}_d)[\mathbb{I}_{p,l}] \subseteq \mathbb{I}_{q,k} \to f[\mathbb{I}_{\frac{p+d}{2},l+1}] \subseteq \mathbb{I}_{q,k}), \tag{InElim}$$
$$\forall_{f,d,p,l,q,k}(f[\mathbb{I}_{\frac{p+d}{2},l+1}] \subseteq \mathbb{I}_{q,k} \to (f \circ \mathrm{In}_d)[\mathbb{I}_{p,l}] \subseteq \mathbb{I}_{q,k}), \tag{InIntro}$$
$$\forall_{f,q}(q \le -\frac{1}{4} \to f[\mathbb{I}] \subseteq \mathbb{I}_{q,2} \to f[\mathbb{I}] \subseteq \mathbb{I}_{\mathsf{L}}), \tag{UcfLeft}$$
$$\forall_{f,q}(-\frac{1}{4} \le q \le \frac{1}{4} \to f[\mathbb{I}] \subseteq \mathbb{I}_{q,2} \to f[\mathbb{I}] \subseteq \mathbb{I}_{\mathsf{M}}), \tag{UcfMiddle}$$
$$\forall_{f,q}(\frac{1}{4} \le q \to f[\mathbb{I}] \subseteq \mathbb{I}_{q,2} \to f[\mathbb{I}] \subseteq \mathbb{I}_{\mathsf{R}}). \tag{UcfRight}$$

Note that the above axioms are all non-computational. The property of Id is specified by (UcfId). Our functions are bounded due to (UcfBound). The more the input given to a function is precise, the same or the more the output is precise due to (UcfInputSucc). For the two operations on functions, our intuition is kept by four axioms, (OutElim), (OutIntro), (InElim) and (InIntro). For example in (OutIntro), assume that the codomain of the function is in $\mathbb{I}_d$, and the image of $\mathbb{I}_{p,l}$ by f is covered by $\mathbb{I}_{\frac{q+d}{2},k+1}$. As $(\mathrm{Out}_d \circ f)(x)$ is intuitively $2f(x) - d$, the image of $\mathbb{I}_{p,l}$ by $(\mathrm{Out}_d \circ f)(x)$ is covered by $\mathbb{I}_{q,k}$

whose center is shifted by $\lambda_x(2x-d)$ and length is two times in comparison with $\mathbb{I}_{\frac{q+d}{2},k+1}$. The last three axioms, (UcfLeft), (UcfMiddle) and (UcfRight) reduce our reasoning on abstract functions to one on rational numbers.

We give definitions of two predicates C and ${}^{\mathrm{co}}\mathrm{Write}$ which are corresponding to the type-1 and the type-0 representations, respectively. First, we define C of arity (ϕ).

Definition 3.5.5 (Uniform continuity of abstract real functions)**.** We define a predicate C of arity (ϕ) to state the uniform continuity of abstract real functions as follows:

$$C := \{f \mid \forall_k^{\mathrm{c}} \exists_l^{\mathrm{d}} \mathrm{B}_{l,k} f\}, \text{ where } \mathrm{B}_{l,k} := \{f \mid \forall_p^{\mathrm{c}} \exists_q^{\mathrm{l}} (f[\mathbb{I}_{p,l}] \subseteq \mathbb{I}_{q,k})\}.$$

Suppose that Cf holds and let k be given to specify the precision of the output of f. Then, there exists l specifying the required precision of the input of f, and also there exists q for any p such that the image of the interval $\mathbb{I}_{p,l}$ is covered by the interval $\mathbb{I}_{q,k}$. According to the realizability interpretation, the extracted term type of the above predicate C is

$$\mathbf{N} \to \mathbf{N} \times (\mathbf{Q} \to \mathbf{Q}),$$

which can be used as a type-1 uniformly continuous functions. It is considered to be a pair of an approximating map h and a modulus of uniform continuity ω. The Cauchy modulus α is fixed to be the identity.

Next, we define an inductive predicate Read_X and a coinductive predicate ${}^{\mathrm{co}}\mathrm{Write}$. Both of them are of arity (ϕ).

Definition 3.5.6 (Read)**.** Let X and P be predicate variables of arity (ϕ). Define an inductive predicate Read_X of arity (ϕ) as follows.

$$\begin{array}{ll}
\forall_f^{\mathrm{nc}} \forall_d^{\mathrm{c}} (f[\mathbb{I}] \subseteq \mathbb{I}_d \to X(\mathrm{Out}_d \circ f) \to \mathrm{Read}_X f), & \mathrm{Read}_0^+ \\
\forall_f^{\mathrm{nc}} (\mathrm{Read}_X(f \circ \mathrm{In_L}) \to \mathrm{Read}_X(f \circ \mathrm{In_M}) \to \mathrm{Read}_X(f \circ \mathrm{In_R}) \to \mathrm{Read}_X f), & \mathrm{Read}_1^+ \\
\forall_f^{\mathrm{nc}} (\mathrm{Read}_X f \to \forall_f^{\mathrm{nc}} \forall_d^{\mathrm{c}} (f[I] \subseteq I_d \to X(\mathrm{Out}_d \circ f) \to Pf) \to & \\
\quad \forall_f^{\mathrm{nc}} (\mathrm{Read}_X(f \circ \mathrm{In_L}) \to P(f \circ \mathrm{In_L}) \to & \\
\quad\quad \mathrm{Read}_X(f \circ \mathrm{In_M}) \to P(f \circ \mathrm{In_M}) \to & \mathrm{Read}^- \\
\quad\quad \mathrm{Read}_X(f \circ \mathrm{In_R}) \to P(f \circ \mathrm{In_R}) \to Pf) \to & \\
\quad Pf). &
\end{array}$$

Definition 3.5.7 (${}^{\mathrm{co}}\mathrm{Write}$)**.** Let Q be a predicate variable of arity ϕ. Define a coinductive predicate ${}^{\mathrm{co}}\mathrm{Write}$ of arity (ϕ) as follows.

$$\begin{array}{ll}
\forall_f^{\mathrm{nc}} ({}^{\mathrm{co}}\mathrm{Write} f \to f \operatorname{eqd} \mathrm{Id} \vee^{\mathrm{r}} \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}} f), & {}^{\mathrm{co}}\mathrm{Write}^- \\
\forall_f^{\mathrm{nc}} (Q\, f \to \forall_f^{\mathrm{nc}} (Q\, f \to f \operatorname{eqd} \mathrm{Id} \vee^{\mathrm{r}} \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee^{\mathrm{d}} Q} f) \to {}^{\mathrm{co}}\mathrm{Write}\, f). & {}^{\mathrm{co}}\mathrm{Write}^+
\end{array}$$

3.5.2 Proofs

We have two main results in this section: Proposition 3.5.11 and Proposition 3.5.14. We start from Proposition 3.5.11 which is dependent on Lemma 3.5.8, Lemma 3.5.9 and Lemma 3.5.10.

Lemma 3.5.8.

1. $\forall_f^{\mathrm{nc}}\forall_d^{\mathrm{c}}\forall_{k,l}(f[\mathbb{I}] \subseteq \mathbb{I}_d \to \mathrm{B}_{l,k+1}f \to \mathrm{B}_{l,k}(\mathrm{Out}_d \circ f))$.

2. $\forall_f^{\mathrm{nc}}\forall_d^{\mathrm{c}}(f[\mathbb{I}] \subseteq \mathbb{I}_d \to \mathrm{C}f \to \mathrm{C}(\mathrm{Out}_d \circ f))$.

3. $\forall_f^{\mathrm{nc}}\forall_d^{\mathrm{c}}\forall_{k,l}(\mathrm{B}_{l+1,k}f \to \mathrm{B}_{l,k}(f \circ \mathrm{In}_d))$.

4. $\forall_f^{\mathrm{nc}}\forall_d^{\mathrm{c}}(\mathrm{C}f \to \mathrm{C}(f \circ \mathrm{In}_d))$.

Proof. 1. Let f, d, k and l be given and assume $f[\mathbb{I}] \subseteq \mathbb{I}_d$ and $\mathrm{B}_{l,k+1}f$, which is by unfolding $\forall_{p\in I}^{\mathrm{c}}\exists_q^{\mathrm{l}}(f[\mathbb{I}_{p,l}] \subseteq \mathbb{I}_{q,k+1})$. We prove $\mathrm{B}_{l,k}(\mathrm{Out}_d \circ f)$, i.e., $\forall_{p\in I}^{\mathrm{c}}\exists_q^{\mathrm{l}}(\mathrm{Out}_d \circ f[\mathbb{I}_{p,l}] \subseteq \mathbb{I}_{q,k})$. Let p be given. Applying $\forall_{p\in I}^{\mathrm{c}}\exists_q^{\mathrm{l}}(f[\mathbb{I}_{p,l}] \subseteq \mathbb{I}_{q,k+1})$ to p, it yields some q_0 such that $f[\mathbb{I}_{p,l}] \subseteq \mathbb{I}_{q_0,k+1}$. To prove the goal $\exists_q^{\mathrm{l}}(\mathrm{Out}_d \circ f[\mathbb{I}_{p,l}] \subseteq \mathbb{I}_{q,k})$, let q be $2q_0 - d$, then it suffices to prove $\mathrm{Out}_d \circ f[\mathbb{I}_{p,l}] \subseteq \mathbb{I}_{2q_0-d,k}$. It is done by applying (OutIntro) to the assumptions $f[\mathbb{I}] \subseteq \mathbb{I}_d$ and $f[\mathbb{I}_{p,l}] \subseteq \mathbb{I}_{q_0,k+1}$.

2. Let f and d be given and assume $f[\mathbb{I}] \subseteq \mathbb{I}_d$ and $\mathrm{C}f$, i.e., $\forall_k^{\mathrm{c}}\exists_l^{\mathrm{d}}\mathrm{B}_{l,k}f$. We prove $\mathrm{C}(\mathrm{Out}_d \circ f)$, i.e., $\forall_k^{\mathrm{c}}\exists_l^{\mathrm{d}}\mathrm{B}_{l,k}(\mathrm{Out}_d \circ f)$. Let k by given and prove $\exists_l^{\mathrm{d}}\mathrm{B}_{l,k}(\mathrm{Out}_d \circ f)$. Applying $\forall_k^{\mathrm{c}}\exists_l^{\mathrm{d}}\mathrm{B}_{l,k}f$ to $k+1$, it further yields l_0 such that $\mathrm{B}_{l_0,k+1}f$. Let l be l_0 in our goal, the by Lemma 3.5.8.1 $\mathrm{B}_{l_0,k}(\mathrm{Out}_d \circ f)$ holds.

3. Let f, d, k and l be given and assume $\mathrm{B}_{l+1,k}f$, i.e., $\forall_{p\in I}^{\mathrm{c}}\exists_q^{\mathrm{l}}(f[\mathbb{I}_{p,l+1}] \subseteq \mathbb{I}_{q,k})$. We prove $\mathrm{B}_{l,k}(f \circ \mathrm{In}_d)$, i.e., $\forall_{p\in I}^{\mathrm{c}}\exists_q^{\mathrm{l}}(f \circ \mathrm{In}_d[\mathbb{I}_{p,l}] \subseteq \mathbb{I}_{q,k})$. Let p be given. Applying $\forall_{p\in I}^{\mathrm{c}}\exists_q^{\mathrm{l}}(f[\mathbb{I}_{p,l+1}] \subseteq \mathbb{I}_{q,k})$ to $\frac{p+d}{2}$, then it further yields q_0 such that $f[\mathbb{I}_{\frac{p+d}{2},l+1}] \subseteq \mathbb{I}_{q_0,k}$. To prove the goal $\exists_q^{\mathrm{l}}(f \circ \mathrm{In}_d[\mathbb{I}_{p,l}] \subseteq \mathbb{I}_{q,k})$, let q be q_0 and use (InIntro).

4. Let f and d be given and assume $\mathrm{C}f$, i.e., $\forall_k^{\mathrm{c}}\exists_l^{\mathrm{d}}\mathrm{B}_{l,k}f$. We prove $\mathrm{C}(f \circ \mathrm{In}_d)$, i.e., $\forall_k^{\mathrm{c}}\exists_l^{\mathrm{d}}\mathrm{B}_{l,k}f \circ \mathrm{In}_d$. Let k be given. Applying $\forall_k^{\mathrm{c}}\exists_l^{\mathrm{d}}\mathrm{B}_{l,k}f$ to k, then it further yields l_0 such that $\mathrm{B}_{l_0,k}f$, i.e., $\forall_{p\in I}^{\mathrm{c}}\exists_q^{\mathrm{l}}(f[\mathbb{I}_{p,l_0}] \subseteq \mathbb{I}_{q,k})$. Now we show that $\mathrm{B}_{l_0+1,k}f \circ \mathrm{In}_d$, i.e., $\forall_{p\in I}^{\mathrm{c}}\exists_q^{\mathrm{l}}(f[\mathbb{I}_{p,l_0+1}] \subseteq \mathbb{I}_{q,k})$, also holds. Let p be given. Applying $\forall_{p\in I}^{\mathrm{c}}\exists_q^{\mathrm{l}}(f[\mathbb{I}_{p,l_0}] \subseteq \mathbb{I}_{q,k})$ to p, it further yields q_0 such that $f[\mathbb{I}_{p,l_0}] \subseteq \mathbb{I}_{q_0,k}$. To prove $\exists_q^{\mathrm{l}}(f[\mathbb{I}_{p,l_0+1}] \subseteq \mathbb{I}_{q,k})$, let q be q_0 and apply (UcfInputSucc) to $f[\mathbb{I}_{p,l_0}] \subseteq \mathbb{I}_{q_0,k}$. For the main goal $\exists_l^{\mathrm{d}}\mathrm{B}_{l,k}f \circ \mathrm{In}_d$, let l be l_0 and we prove $\mathrm{B}_{l_0,k}f \circ \mathrm{In}_d$, which is implied by Lemma 3.5.8.3 and $\mathrm{B}_{l_0+1,k}f$.

□

Lemma 3.5.9. $\forall_f^{\mathrm{nc}}(\mathrm{B}_{0,2}f \to \exists_d^{\mathrm{l}}(f[\mathbb{I}] \subseteq \mathbb{I}_d))$.

Proof. Let f be given and assume $\mathrm{B}_{0,2}f$ which implies $\exists^{\mathrm{l}}_q(f[\mathbb{I}_{0,0}] \subseteq \mathbb{I}_{q,2})$ from the definition. Hence for some q, $f[\mathbb{I}_{0,0}] \subseteq \mathbb{I}_{q,2}$ holds. We prove $\exists^{\mathrm{l}}_d(f[\mathbb{I}] \subseteq \mathbb{I}_d)$ by finding a d such that $f[\mathbb{I}] \subseteq \mathbb{I}_d$ holds. By case distinction on q, either $q \leq -\frac{1}{4}$, $-\frac{1}{4} \leq q \leq \frac{1}{4}$ or $\frac{1}{4} \leq q$ holds. For each case we respectively choose L, M or R for d. This d satisfies $f[\mathbb{I}] \subseteq \mathbb{I}_d$ due to the corresponding axiom, (UcfLeft), (UcfMiddle) or (UcfRight). □

Lemma 3.5.9 determines a signed digit which approximates the output of uniformly continuous functions. Suppose that we input the interval $\mathbb{I}$ to an abstract function and the length of its output interval is less than $\frac{1}{2}$. Then, the output of the abstract function is either in $\mathbb{I}_{\mathsf{L}}$, $\mathbb{I}_{\mathsf{M}}$ or $\mathbb{I}_{\mathsf{R}}$.

Lemma 3.5.10. $\forall^{\mathrm{c}}_l \forall^{\mathrm{nc}}_f(\mathrm{B}_{l,2}f \to \mathrm{C}f \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee \mathrm{C}}f)$.

Proof. By induction on l. *Base:* $l = 0$. Let f be given and assume $\mathrm{B}_{0,2}f$ and $\mathrm{C}f$. We prove $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee \mathrm{C}}f$ by means of Read^+_0. It suffices to prove $f[\mathbb{I}] \subseteq \mathbb{I}_d$ and $\mathrm{C}(\mathrm{Out}_d \circ f)$ for some d. By Lemma 3.5.9, there is some d with $f[\mathbb{I}] \subseteq \mathbb{I}_d$ for the first one. Moreover, $\mathrm{C}f$ implies $\mathrm{C}(\mathrm{Out}_d \circ f)$ provided $f[\mathbb{I}] \subseteq \mathbb{I}_d$ by Lemma 3.5.8.2, hence $({}^{\mathrm{co}}\mathrm{Write} \vee^{\mathrm{d}} \mathrm{C})(\mathrm{Out}_d \circ f)$ as desired. *Step:* $l \mapsto l + 1$. Assume the following induction hypothesis

$$\forall^{\mathrm{nc}}_f(\mathrm{B}_{l,2}f \to \mathrm{C}f \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee \mathrm{C}}f), \tag{3.5}$$

and prove $\forall^{\mathrm{nc}}_f(\mathrm{B}_{l+1,2}f \to \mathrm{C}f \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee \mathrm{C}}f)$ by means of the introduction axiom Read^+_1. It suffices to prove $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee \mathrm{C}}f \circ \mathrm{In}_d$ for all d. Let f be given and assume $\mathrm{B}_{l+1,2}f$ and $\mathrm{C}f$. By Lemma 3.5.8.3 and Lemma 3.5.8.4, which imply $\mathrm{B}_{l,2}f \circ \mathrm{In}_d$ and $\mathrm{C}f \circ \mathrm{In}_d$ for all d. The induction hypothesis (3.5) yields $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee \mathrm{C}}f \circ \mathrm{In}_d$ for all d as desired. □

Proposition 3.5.11 (Type-1 u.c.f. to type-0 u.c.f.)**.** $\forall^{\mathrm{nc}}_f(\mathrm{C}f \to {}^{\mathrm{co}}\mathrm{Write}f)$.

Proof. Let f be given and assume $\mathrm{C}f$. We prove ${}^{\mathrm{co}}\mathrm{Write}f$ by the greatest fixed point axiom. It suffices to prove $\forall^{\mathrm{nc}}_f(\mathrm{C}f \to f \operatorname{eqd} \mathrm{Id} \vee^{\mathrm{r}} \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee^{\mathrm{d}} \mathrm{C}}f)$. Let f be given and $\mathrm{C}f$, which implies $\exists^{\mathrm{d}}_l \mathrm{B}_{l,2}f$. By Lemma 3.5.10, $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee^{\mathrm{d}} \mathrm{C}}f$ holds. □

Next, we prove Proposition 3.5.14, i.e., the opposite direction of Proposition 3.5.11. The following two lemmas are needed.

Lemma 3.5.12. $\forall^{\mathrm{nc}}_{f,k,l} \forall^{\mathrm{c}}_d(f[\mathbb{I}] \subseteq \mathbb{I}_d \to \mathrm{B}_{l,k}(\mathrm{Out}_d \circ f) \to \mathrm{B}_{l,k+1}f)$.

Proof. Let f, k, l and d be given and assume $f[\mathbb{I}] \subseteq \mathbb{I}_d$ and

$$\mathrm{B}_{l,k}(\mathrm{Out}_d \circ f). \tag{3.6}$$

We prove $\mathrm{B}_{l,k+1}f$, i.e., $\forall^{\mathrm{c}}_{p \in I} \exists^{\mathrm{l}}_q(f[\mathbb{I}_{p,l}] \subseteq \mathbb{I}_{q,k+1})$. Let p_0 be given such that $\exists^{\mathrm{l}}_q(f[\mathbb{I}_{p_0,l}] \subseteq \mathbb{I}_{q,k+1})$. Our goal is to find a q such as $f[\mathbb{I}_{p_0,l}] \subseteq \mathbb{I}_{q_0,k+1}$. The assumption (3.6), i.e., $\forall^{\mathrm{c}}_{p \in I} \exists^{\mathrm{l}}_q((\mathrm{Out}_d \circ f)[\mathbb{I}_{p,l}] \subseteq \mathbb{I}_{q,k})$, implies $\exists^{\mathrm{l}}_q((\mathrm{Out}_d \circ f)[\mathbb{I}_{2p_0-d,l}] \subseteq \mathbb{I}_{q,k})$. Hence there is some q_0 such that $(\mathrm{Out}_d \circ f)[\mathbb{I}_{2p_0-d,l}] \subseteq \mathbb{I}_{q_0,k}$. For this q_0, we can derive $f[\mathbb{I}_{p_0,l}] \subseteq \mathbb{I}_{q_0,k+1}$ by means of (OutElim). □

Lemma 3.5.13. *We prove*

$$\forall^{\mathrm{nc}}_{f,k,l_{\mathsf{L}},l_{\mathsf{M}},l_{\mathsf{R}}}(\mathrm{B}_{l_{\mathsf{L}},k+1}(f\circ\mathrm{In}_{\mathsf{L}})\to\mathrm{B}_{l_{\mathsf{M}},k+1}(f\circ\mathrm{In}_{\mathsf{M}})\to\mathrm{B}_{l_{\mathsf{R}},k+1}(f\circ\mathrm{In}_{\mathsf{R}})\to\mathrm{B}_{l,k+1}f),$$

where $l:=\max\{l_{\mathsf{L}},l_{\mathsf{M}},l_{\mathsf{R}}\}+1$.

Proof. Let f, k, l_{L}, l_{M} and l_{R} be given and assume $\mathrm{B}_{l_{\mathsf{L}},k+1}(f\circ\mathrm{In}_{\mathsf{L}})$, $\mathrm{B}_{l_{\mathsf{M}},k+1}(f\circ\mathrm{In}_{\mathsf{M}})$ and $\mathrm{B}_{l_{\mathsf{R}},k+1}(f\circ\mathrm{In}_{\mathsf{R}})$. We prove $\mathrm{B}_{l,k+1}f$, i.e., $\forall^{\mathrm{c}}_p\exists^{\mathrm{l}}_q(f[\mathbb{I}_{p,l}]\subseteq\mathbb{I}_{q,k+1})$. Let p_0 be given, then our goal is $\exists^{\mathrm{l}}_q(f[\mathbb{I}_{p_0,l}]\subseteq\mathbb{I}_{q,k+1})$. By case distinction on p_0, either of $p_0\le-\frac{1}{4}$, $-\frac{1}{4}\le p_0\le\frac{1}{4}$ or $\frac{1}{4}\le p_0$ holds. For each case we let d be L, M or R, respectively. For any d we can proceed the following to prove the goal. The assumption $\mathrm{B}_{l_d,k+1}(f\circ\mathrm{In}_d)$ i.e., $\forall^{\mathrm{c}}_p\exists^{\mathrm{l}}_q((f\circ\mathrm{In}_d)[\mathbb{I}_{p,l_d}]\subseteq\mathbb{I}_{q,k+1})$ implies $(f\circ\mathrm{In}_d)[\mathbb{I}_{2p_0-d,l_d}]\subseteq\mathbb{I}_{q_0,k+1}$ for some q_0. By (InElim), $f[\mathbb{I}_{p_0,l_d+1}]\subseteq\mathbb{I}_{q_0,k+1}$ holds. For this q_0 we can prove the goal $f[\mathbb{I}_{p_0,l}]\subseteq\mathbb{I}_{q_0,k+1}$ because of $l\ge l_0+1$ and (UcfInputSucc). □

Proposition 3.5.14 (Type-0 u.c.f. into type-1 u.c.f.)**.** $\forall^{\mathrm{nc}}_f({}^{\mathrm{co}}\mathrm{Write}f\to\mathrm{C}f)$.

Proof. It suffices to prove $\forall^{\mathrm{c}}_k\forall^{\mathrm{nc}}_f({}^{\mathrm{co}}\mathrm{Write}f\to\exists^{\mathrm{d}}_l\mathrm{B}_{l,k}f)$ by induction on k. *Case* $k=0$. Our goal is $\forall^{\mathrm{nc}}_f({}^{\mathrm{co}}\mathrm{Write}f\to\exists^{\mathrm{d}}_l\mathrm{B}_{l,0}f)$. Let f be given and ${}^{\mathrm{co}}\mathrm{Write}f$. Let l be 0 and prove $\mathrm{B}_{0,0}f$, i.e., $\forall^{\mathrm{c}}_p\exists^{\mathrm{l}}_q f[\mathbb{I}_{p,0}]\subseteq\mathbb{I}_{q,0}$. Let p be given q be 0. From the axiom, $f[\mathbb{I}_{p,0}]\subseteq\mathbb{I}_{0,0}$ trivially holds. *Case* $k\mapsto k+1$. Our induction hypothesis is

$$\forall^{\mathrm{nc}}_f({}^{\mathrm{co}}\mathrm{Write}f\to\exists^{\mathrm{d}}_l\mathrm{B}_{l,k}f). \tag{3.7}$$

Our goal is $\forall^{\mathrm{nc}}_f({}^{\mathrm{co}}\mathrm{Write}f\to\exists^{\mathrm{d}}_l\mathrm{B}_{l,k+1}f)$. Let f be given and assume ${}^{\mathrm{co}}\mathrm{Write}f$. We prove $\exists^{\mathrm{d}}_l\mathrm{B}_{l,k+1}f$. Using ${}^{\mathrm{co}}\mathrm{Write}^+$ and our assumption ${}^{\mathrm{co}}\mathrm{Write}f$, the following holds.

$$f\operatorname{eqd}\mathrm{Id}\vee\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}f.$$

Consider two cases from the disjunction. *Left case.* Assume $f\operatorname{eqd}\mathrm{Id}$. We prove $\exists^{\mathrm{d}}_l\mathrm{B}_{l,k+1}\mathrm{Id}$, i.e., $\exists^{\mathrm{d}}_l\forall^{\mathrm{c}}_p\exists^{\mathrm{l}}_q(\mathrm{Id}[\mathbb{I}_{p,l}]\subseteq\mathbb{I}_{q,k+1})$. Let l be $k+1$ and assume p, and also let q be p, then it suffices to prove $\mathrm{Id}[\mathbb{I}_{p,k+1}]\subseteq\mathbb{I}_{p,k+1}$, which is direct from (UcfId). *Right case.* Assume $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}f$ and prove $\exists^{\mathrm{d}}_l\mathrm{B}_{l,k+1}f$. By using Read^- for $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}f$, we have the following two new goals.

$$\forall^{\mathrm{nc}}_f\forall^{\mathrm{c}}_d(f[\mathbb{I}]\subseteq\mathbb{I}_d\to{}^{\mathrm{co}}\mathrm{Write}(\mathrm{Out}_d\circ f)\to\exists^{\mathrm{d}}_l\mathrm{B}_{l,k+1}f)$$

and

$$\begin{aligned}&\forall^{\mathrm{nc}}_f(\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}(f\circ\mathrm{In}_{\mathsf{L}})\to\exists^{\mathrm{d}}_l\mathrm{B}_{l,k+1}(f\circ\mathrm{In}_{\mathsf{L}})\to\\&\quad\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}(f\circ\mathrm{In}_{\mathsf{M}})\to\exists^{\mathrm{d}}_l\mathrm{B}_{l,k+1}(f\circ\mathrm{In}_{\mathsf{M}})\to\\&\quad\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}(f\circ\mathrm{In}_{\mathsf{R}})\to\exists^{\mathrm{d}}_l\mathrm{B}_{l,k+1}(f\circ\mathrm{In}_{\mathsf{R}})\to\exists^{\mathrm{d}}_l\mathrm{B}_{l,k+1}f)\end{aligned}$$

For the former one, let f and d be given and assume $f[\mathbb{I}]\subseteq\mathbb{I}_d$ and ${}^{\mathrm{co}}\mathrm{Write}(\mathrm{Out}_d\circ f)$. By our induction hypothesis (3.7) and ${}^{\mathrm{co}}\mathrm{Write}(\mathrm{Out}_d\circ f)$, $\exists^{\mathrm{d}}_l\mathrm{B}_{l,k}(\mathrm{Out}_d\circ f)$ holds, i.e., there is some l such that $\mathrm{B}_{l,k}(\mathrm{Out}_d\circ f)$. It implies $\mathrm{B}_{l,k+1}f$ by Lemma 3.5.12 as desired. For the latter one, let f be given and assume $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}(f\circ\mathrm{In}_{\mathsf{L}})$, $\exists^{\mathrm{d}}_l\mathrm{B}_{l,k+1}(f\circ\mathrm{In}_{\mathsf{L}})$, $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}(f\circ\mathrm{In}_{\mathsf{M}})$, $\exists^{\mathrm{d}}_l\mathrm{B}_{l,k+1}(f\circ\mathrm{In}_{\mathsf{M}})$, $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}(f\circ\mathrm{In}_{\mathsf{R}})$ and $\exists^{\mathrm{d}}_l\mathrm{B}_{l,k+1}(f\circ\mathrm{In}_{\mathsf{R}})$. We prove $\exists^{\mathrm{d}}_l\mathrm{B}_{l,k+1}f$. From assumptions, there are l_{L}, l_{M} and l_{R} such that $\mathrm{B}_{l_{\mathsf{L}},k+1}(f\circ\mathrm{In}_{\mathsf{L}})$, $\mathrm{B}_{l_{\mathsf{M}},k+1}(f\circ\mathrm{In}_{\mathsf{M}})$ and $\mathrm{B}_{l_{\mathsf{R}},k+1}(f\circ\mathrm{In}_{\mathsf{R}})$. By Lemma 3.5.13, $\mathrm{B}_{l,k+1}f$ holds for $l:=\max\{l_{\mathsf{L}},l_{\mathsf{M}},l_{\mathsf{R}}\}+1$ as desired. □

3.5.3 Program Extraction

We study the computational contents of the proofs by program extraction. From Proposition 3.5.11 the term `ucf1to0` shown in Figure 3.11 is extracted. This program corecursively

$$\lambda_g({}^{\mathrm{co}}\mathcal{R}_{\mathbf{W}}^{\mathbf{N}\to\mathbf{N}\times(\mathbf{Q}\to\mathbf{Q})}\, g\, \lambda_{g_0}(\mathrm{Case}\ g_0(2)\ \mathrm{of}\ \langle k,h\rangle \to \mathtt{L_{10}}\, k\, g_0\, h))$$

Figure 3.11: $\mathtt{ucf1to0}^{(\mathbf{N}\to\mathbf{N}\times(\mathbf{Q}\to\mathbf{Q}))\to\mathbf{W}}$

constructs a $\mathbf{W}$-cototal $\mathbf{R}_{\mathbf{W}}$-total ideal by stacking $\mathbf{R}_{\mathbf{W}}$-total ideals computed by $\mathtt{L_{10}}$, which is extracted from Lemma 3.5.10 as shown in Figure 3.12. The modulus of continuity k at

$$\begin{aligned}\lambda_k(\mathcal{R}_{\mathbf{N}}\, k\ \lambda_{h,g}(&\mathsf{Put}\,(\mathtt{L_9}h)\,(\mathsf{InR}(\mathtt{L_{8ii}}(\mathtt{L_9}h)g)))\\ &\lambda_{l,g_0,h,g_1}(\mathsf{Get}\,(g_0(\mathtt{L_{8iii}}\,\mathsf{L}\,h)(\mathtt{L_{8iv}}\,\mathsf{L}\,g_1))\\ &\qquad(g_0(\mathtt{L_{8iii}}\,\mathsf{M}\,h)(\mathtt{L_{8iv}}\,\mathsf{M}\,g_1))\\ &\qquad(g_0(\mathtt{L_{8iii}}\,\mathsf{R}\,h)(\mathtt{L_{8iv}}\,\mathsf{R}\,g_1))))\end{aligned}$$

Figure 3.12: $\mathtt{L_{10}}^{\mathbf{N}\to(\mathbf{Q}\to\mathbf{Q})\to(\mathbf{N}\to\mathbf{N}\times(\mathbf{Q}\to\mathbf{Q}))\to\mathbf{R}_{\mathbf{W}+(\mathbf{N}\to\mathbf{N}\times(\mathbf{Q}\to\mathbf{Q}))}}$

precision 2 is fed to $\mathtt{L_{10}}$, then an approximation of h as an $\mathbf{R}_{\mathbf{W}}$-total ideal is computed. In fact, k is the number of input signed digits to be read to determine one output signed digit.

The extracted term $\mathtt{L_{10}}$ takes as input a natural number k, a rational function g and a function $h : \mathbf{N} \to \mathbf{N} \times (\mathbf{Q} \to \mathbf{Q})$ (in our application, we only call $\mathtt{L_{10}}$ with $\langle n, w\rangle = h\, 2$). Using recursion over n, it computes an approximation of h by a complete tree of height n with 3^n leaves – an $\mathbf{R}_{\mathbf{W}+(\mathbf{N}\to\mathbf{N}\times(\mathbf{Q}\to\mathbf{Q}))}$-total ideal. At the leaves, a signed digit d – computed from w using $\mathtt{L_9}$ in Figure 3.13 – and the remainder of the approximation of h – computed by $\mathtt{L_{8ii}}$ below, using d – is stored. At internal branching nodes, we split the domain of h

$$\begin{aligned}\lambda_g(&\mathbf{if}\ g(0)+\tfrac{1}{4}>0\ \mathbf{then\ if}\ g(0)-\tfrac{1}{4}>0\ \mathbf{then}\ \mathsf{R}\\ &\qquad\mathbf{else}\ \mathsf{M}\ \mathbf{else}\ \mathsf{L})\end{aligned}$$

Figure 3.13: $\mathtt{L_9}^{(\mathbf{Q}\to\mathbf{Q})\to\mathbf{SD}}$

into three subdomains – left, middle and right – modify w and h accordingly (using $\mathtt{L_{8iii}}$ and $\mathtt{L_{8iv}}$ below), and recurse.

There are several more programs extracted from lemmas. From Lemma 3.5.9 we extract $\mathtt{L_9}$ which is similar to `L` we extracted from Lemma 3.3.8. This program determines a d such that the whole values of the given g is covered by $\mathbb{I}_d$. In our extracted program, it is used

under the assumption that the values of g given to $\mathtt{L_9}$ is covered by an interval of length $\frac{1}{2}$. Therefore, it suffices to check the value of g at 0. The terms in Figure 3.14 are extracted from each of Lemma 3.5.8. Next, we consider the extracted program from Proposition 3.5.14

$$
\begin{aligned}
&\mathtt{L_{8i}}^{\mathbf{SD}\to(\mathbf{Q}\to\mathbf{Q})\to(\mathbf{Q}\to\mathbf{Q})}\\
&\mathtt{L_{8i}} \coloneqq \lambda_{d,h}(\lambda_a(2h(a)-d))\\
&\mathtt{L_{8ii}}^{\mathbf{SD}\to(\mathbf{N}\to\mathbf{N}\times(\mathbf{Q}\to\mathbf{Q}))\to(\mathbf{N}\to\mathbf{N}\times(\mathbf{Q}\to\mathbf{Q}))}\\
&\mathtt{L_{8ii}} \coloneqq \lambda_{d,g,k}(\text{Case } g(k+1) \text{ of } \langle l,h\rangle \to \langle l, \mathtt{L_{8i}}\, d\, h\rangle)\\
&\mathtt{L_{8iii}}^{\mathbf{SD}\to(\mathbf{Q}\to\mathbf{Q})\to(\mathbf{Q}\to\mathbf{Q})}\\
&\mathtt{L_{8iii}} \coloneqq \lambda_{d,h}(\lambda_a(h(\frac{a+d}{2})))\\
&\mathtt{L_{8iv}}^{\mathbf{SD}\to(\mathbf{N}\to\mathbf{N}\times(\mathbf{Q}\to\mathbf{Q}))\to(\mathbf{N}\to\mathbf{N}\times(\mathbf{Q}\to\mathbf{Q}))}\\
&\mathtt{L_{8iv}} \coloneqq \lambda_{d,g,k}(\text{Case } g(k) \text{ of } \langle l,h\rangle \to (\text{Case } l \text{ of } \mathsf{0} \to \langle \mathsf{0}, \mathtt{L_{8iii}}\, d\, h\rangle\\
&\qquad\qquad\qquad\qquad\qquad\qquad \mathsf{S}\, n \to \langle n, \mathtt{L_{8iii}}\, d\, h\rangle))
\end{aligned}
$$

Figure 3.14: Programs extracted from Lemma 3.5.8

shown in Figure 3.15. Let f be a name of a variable ranging over $\mathbf{W}\to\mathbf{N}\times(\mathbf{Q}\to\mathbf{Q})$,

$$
\begin{aligned}
&\lambda_{w,n}(\mathcal{R}_{\mathbf{N}}^{\mathbf{W}\to\mathbf{N}\times(\mathbf{N}\to\mathbf{Q})}\, n\\
&\quad \lambda_{\_}\langle \mathsf{0}, \lambda_a 0\rangle\\
&\quad \lambda_{n_0,f,w_0}(\text{Case } \mathcal{D}\, w_0 \text{ of}\\
&\qquad\qquad \lambda_{\_}\langle \mathsf{S}\, n_0, \lambda_a a\rangle\\
&\qquad\qquad \lambda_r(\mathcal{R}_{\mathbf{R_W}}^{\mathbf{N}\times(\mathbf{N}\to\mathbf{Q})}\, r\, \lambda_{d,w_1}(\text{Case } f w_1 \text{ of } \lambda_{m,g}\langle m, \mathtt{L_{12}}\, d\, g\rangle)\\
&\qquad\qquad\qquad\qquad \lambda_{\_,q_\mathsf{L},\_,q_\mathsf{M},\_,q_\mathsf{R}}(\text{Case } q_\mathsf{L} \text{ of}\\
&\qquad\qquad\qquad\qquad\qquad\qquad \lambda_{n_\mathsf{L},g_\mathsf{L}}\text{Case } q_\mathsf{M} \text{ of}\\
&\qquad\qquad\qquad\qquad\qquad\qquad \lambda_{n_\mathsf{M},g_\mathsf{M}}\text{Case } q_\mathsf{R} \text{ of}\\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad \lambda_{n_\mathsf{R},g_\mathsf{R}}\langle \mathsf{S}(\max\, n_\mathsf{L}\, n_\mathsf{M}\, n_\mathsf{R}), \mathtt{L_{13}}\, g_\mathsf{L}\, g_\mathsf{M}\, g_\mathsf{R}\rangle))))\\
&\quad w)
\end{aligned}
$$

Figure 3.15: $\mathtt{ucf0to1}^{\mathbf{W}\to\mathbf{N}\to\mathbf{N}\times(\mathbf{Q}\to\mathbf{Q})}$

q ranging over $\mathbf{N}\times(\mathbf{Q}\to\mathbf{Q})$ and g ranging over $\mathbf{Q}\to\mathbf{Q}$. This program computes a uniformly continuous function of type-1 from one of type-0. The second argument n of type $\mathbf{N}$ specifies the precision of the rational approximation of the function. If n is $\mathsf{0}$, the result is a pair of $\mathsf{0}$ and a function whose value is constantly 0. It trivially satisfies the required

precision; the error of the output has to be less than 2^0 for any input with the error less than 2^0. If n is $\mathsf{S}n_0$, we assume that n_0, the previous result f of the precision n_0. It is going to compute a pair of $\mathbf{N}$ and $\mathbf{Q} \to \mathbf{Q}$ with the precision $\mathsf{S}n_0$ for the given input w_0. There are two cases to consider depending on w_0. If $\mathcal{D}w_0$ is $\mathsf{InL}\,\mathsf{U}$, the expected outcome is the identity function with the precision $\mathsf{S}n_0$, i.e., the pair of $\mathsf{S}n_0$ and $\lambda_a a$. If $\mathcal{D}w_0$ is $\mathsf{InR}\,r$, it inspects r by side recursion. If r is $\mathsf{Put}\,d\,w_1$, we apply f to w_1, which is the result for w_1 of the precision n_0. Since d tells us where to go to get a better precision, it shifts the result by using $\mathtt{L}_{12}$ which is extracted from Lemma 3.5.12 and shown in Figure 3.16. If r is

$$\lambda_{d,g,a}(\tfrac{g(a)+d}{2})$$

Figure 3.16: $\mathtt{L}_{12}{}^{\mathbf{SD}\to(\mathbf{Q}\to\mathbf{Q})\to\mathbf{Q}\to\mathbf{Q}}$

$\mathsf{Get}\,r_{\mathsf{L}}\,r_{\mathsf{M}}\,r_{\mathsf{R}}$, there are three previous results q_{L}, q_{M} and q_{R} available. It takes components n_d and g_d of each q_d and computes the maximum of n_{L}, n_{M} and n_{R}. We require one more precision of this maximum because we read one signed digit from the input at Get. The other component of the pair is the composition of g_{L}, g_{M} and g_{R} by $\mathtt{L}_{13}$ which is extracted from Lemma 3.5.13 shown in Figure 3.17. For the input rational a, it checks which basic

$$\lambda_{g_{\mathsf{L}},g_{\mathsf{M}},g_{\mathsf{R}},a}(\mathbf{if}\ (a \le -\frac{1}{4})\ \mathbf{then}\ g_{\mathsf{L}}(2a+1)\ \mathbf{else\ if}\ (-\frac{1}{4} \le a \le \frac{1}{4})\ \mathbf{then}\ g_{\mathsf{M}}(a)\ \mathbf{else}\ g(2a-1))$$

Figure 3.17: $\mathtt{L}_{13}{}^{(\mathbf{Q}\to\mathbf{Q})\to(\mathbf{Q}\to\mathbf{Q})\to(\mathbf{Q}\to\mathbf{Q})\to\mathbf{Q}\to\mathbf{Q}}$

interval covers this a and selects the appropriate g_d.

Remark 3.5.15. Note that there is a slight difference between the extracted programs in Minlog and ones presented in this section. In the case study in Minlog, there is an algebra `algB` which is defined to be $\mu_\xi((\mathbf{Q} \to \mathbf{Q}) \to \xi)$. Since there is no feature in Minlog to name a comprehension term, we defined $B_{l,k}$ of Definition 3.5.5 as an inductively defined predicate in order to avoid writing a long formula repeatedly. We omit the occurrences of `algB` in order to focus on algorithmic contents.

3.5.4 Experiments

From a type-1 uniformly continuous function f, we compute a type-0 uniformly continuous function by means of our extracted program. We define f by $\langle h, \alpha, \omega\rangle$ where $h\,a\,n := -a$, $\alpha\,n := 0$ and $\omega\,n := n+1$. The input of `type1to0` is $\lambda_n\langle \omega n, \lambda_a(ha(\alpha n))\rangle$ whose type is $\mathbf{N} \to \mathbf{N} \times (\mathbf{Q} \to \mathbf{Q})$. The output is graphically presented in Figure 3.18, where Cont is

omitted, Get is a branching node and Put d is denoted by $-$, 0 or $+$ respectively for $d = \mathsf{L}, \mathsf{M}$ or R.

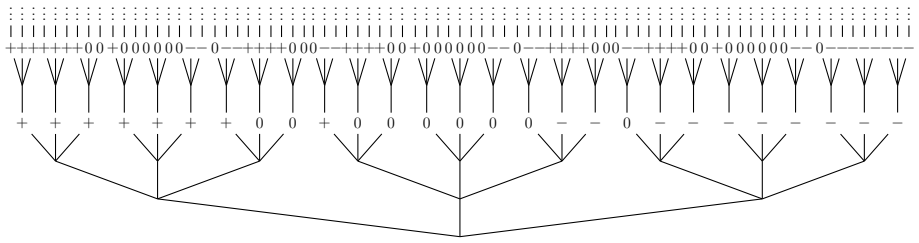

Figure 3.18: Type-0 representation of $f(x) = -x$.

We try the opposite direction, i.e., from type-0 to type-1. Applying `type0to1` to the tree t from the first example, we obtain a function of type $\mathbf{N} \to \mathbf{N} \times (\mathbf{Q} \to \mathbf{Q})$. Giving some k to the argument of the outcome, it results in a pair of a natural number l and a rational function g. For the experiment we give $k = 3$, then $l = 4$ and the function g is a step function as presented in Figure 3.19. It is observable that if the error of the input is

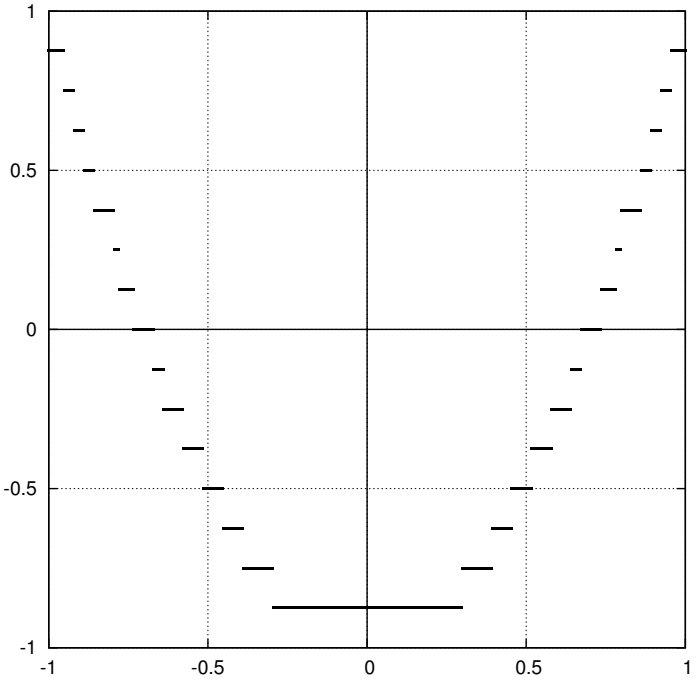

Figure 3.19: Type-1 representation of $f(x) = 2x^2 - 1$ approximated by 3.

less than 2^{-4}, the output approximates $f(x) = 2x^2 - 1$ with the error less than 2^{-3}.

3.6 Application of Uniformly Continuous Functions

We study application of a type-0 uniformly continuous function to a type-0 real number. We extract a program which takes a type-0 uniformly continuous function f and a type-0 real number x, and computes the application $f(x)$ in type-0 real numbers.

For given w and ds of type $\mathbf{W}$ and $\mathbf{L_{SD}}$, the application builds another SDS to output. In order to illustrate an intuition, we focus on Get and Put nodes of w, so that we regard w simply as a tree with 1-branching nodes labelled with a signed digit and 3-branching nodes. The application procedure takes a look at the root of w. If it is a 1-branching node, it outputs the label of this node and goes on with the subtree. If it is a 3-branching node, it reads the head of the input SDS, say d, and chooses the left, the middle, or the right branch corresponding to d. The procedure goes on with the chosen subtree and the tail of the input SDS.

3.6.1 Definitions

The abstract theory of the function application is based on the abstract theory of real numbers in Section 3.3 and the abstract theory of uniformly continuous functions in Section 3.5. We define a constant Apply for the abstract function application additionally to the abstract theories from the previous sections.

Definition 3.6.1 (Apply)**.** Recall that ρ and ϕ are used for abstract types of real numbers and uniformly continuous functions, respectively. Apply is a constant of type $\rho \to \phi \to \rho$. Apply $f\ x$ is abbreviated as $f(x)$.

Then, we enrich the axiom to talk about the function application.

Axiom 3.6.2.

$$\forall_x(\mathrm{Id}(x) \operatorname{eqd} x), \tag{AppId}$$
$$\forall_{f,x,d}(f[\mathbb{I}] \subseteq \mathbb{I}_d \to \mathrm{Va}_d(f(x)) \operatorname{eqd} (\mathrm{Out}_d \circ f)(x)), \tag{VaOut}$$
$$\forall_{f,x,d}(f(\mathrm{Av}_d x) \operatorname{eqd} (f \circ \mathrm{In}_d)(x)), \tag{AvIn}$$
$$\forall_{f,x,q,k}(f[\mathbb{I}] \subseteq \mathbb{I}_{q,k} \to f(x) \in \mathbb{I}_{q,k}). \tag{AppSubElem}$$

The constant Id works as we commonly expect due to (AppId). The operations Va_d and Out_d can replace each other by (VaOut), and the same happens to Av_d and In_d by (AvIn). The relation $\subseteq$ can be replaced by $\in$ due to (AppSubElem).

The following equality between two real numbers is also used.

Axiom 3.6.3.

$$\forall_{x,d}(x \operatorname{eqd} \mathrm{Va}_d(\mathrm{Av}_d x)). \tag{VaAvIdent}$$

We use the predicates ${}^{\mathrm{co}}I$, Read and ${}^{\mathrm{co}}$Write which are defined in Definition 3.3.6, Definition 3.5.6 and Definition 3.5.7, respectively.

3.6.2 Proofs

We prove the following proposition in order to extract a program of the function application.

Proposition 3.6.4. $\forall^{\mathrm{nc}}_{f,x}({}^{\mathrm{co}}\mathrm{Write}f \to {}^{\mathrm{co}}Ix \to {}^{\mathrm{co}}I(f(x)))$.

Proof. Let f and x be given, and assume ${}^{\mathrm{co}}\mathrm{Write}f$ and ${}^{\mathrm{co}}Ix$. We use ${}^{\mathrm{co}}I^{+}$ with a competitor predicate $P = \{z \mid \exists^{\mathrm{r}}_{f,x}({}^{\mathrm{co}}\mathrm{Write}f \wedge^{\mathrm{d}} {}^{\mathrm{co}}Ix \wedge^{\mathrm{l}} z \operatorname{eqd} f(x))\}$ to prove ${}^{\mathrm{co}}I(f(x))$. Since the first premise $P(f(x))$ is clear from the assumptions, it suffices to prove the costep formula $\forall^{\mathrm{nc}}_{z}(Pz \to z \operatorname{eqd} \mathrm{Z} \vee^{\mathrm{r}} \exists^{\mathrm{r}}_{y}\exists^{\mathrm{d}}_{d}(({}^{\mathrm{co}}Iy \vee^{\mathrm{d}} Py) \wedge^{\mathrm{l}} z \operatorname{eqd} \mathrm{Av}_d(y)))$. Let z be given and assume Pz. Unfolding Pz, there are some f_0 and x_0 such that ${}^{\mathrm{co}}\mathrm{Write}f_0$, ${}^{\mathrm{co}}Ix_0$ and $z \operatorname{eqd} f_0(x_0)$ hold. Our goal is $z \operatorname{eqd} \mathrm{Z} \vee^{\mathrm{r}} \exists^{\mathrm{r}}_{y}\exists^{\mathrm{d}}_{d}(({}^{\mathrm{co}}Iy \vee^{\mathrm{d}} Py) \wedge^{\mathrm{l}} z \operatorname{eqd} \mathrm{Av}_d(y))$. Applying ${}^{\mathrm{co}}\mathrm{Write}^{-}$ to ${}^{\mathrm{co}}\mathrm{Write}f_0$, $f_0 \operatorname{eqd} \mathrm{Id} \vee^{\mathrm{r}} \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}f_0$ makes two cases. *Left case.* Assume $f_0 \operatorname{eqd} \mathrm{Id}$. Applying ${}^{\mathrm{co}}I^{-}$ to ${}^{\mathrm{co}}Ix_0$, $x_0 \operatorname{eqd} \mathrm{Z} \vee^{\mathrm{r}} \exists^{\mathrm{r}}_{y_0}\exists^{\mathrm{d}}_{d_0}({}^{\mathrm{co}}Iy_0 \wedge^{\mathrm{l}} x_0 \operatorname{eqd} \mathrm{Av}_{d_0}(y_0))$ makes two subcases. Each side case comes by (AppId). *Right case.* Assume $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}f_0$. We also have ${}^{\mathrm{co}}Ix_0$ and $z \operatorname{eqd} f_0(x_0)$, hence Lemma 3.6.5 applies. □

The Lemma 3.6.5 below is responsible for determining an output signed digit by recursion on an $\mathbf{R_W}$-total tree.

Lemma 3.6.5. *Let* $P := \{z \mid \exists^{\mathrm{r}}_{f,x}({}^{\mathrm{co}}\mathrm{Write}f \wedge^{\mathrm{d}} {}^{\mathrm{co}}Ix \wedge^{\mathrm{l}} z \operatorname{eqd} f(x))\}$, *then*

$$\forall^{\mathrm{nc}}_{f,x}(\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}f \to {}^{\mathrm{co}}Ix \to f(x) \operatorname{eqd} \mathrm{Z} \vee^{\mathrm{r}} \exists^{\mathrm{r}}_{y}\exists^{\mathrm{d}}_{d}(({}^{\mathrm{co}}Iy \vee^{\mathrm{d}} Py) \wedge^{\mathrm{l}} f(x) \operatorname{eqd} \mathrm{Av}_d(y))).$$

Proof. We prove the following formula which is same as the proposition.

$$\forall^{\mathrm{nc}}_{f}(\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}\, f \to \forall^{\mathrm{nc}}_{x}({}^{\mathrm{co}}Ix \to f(x) \operatorname{eqd} \mathrm{Z} \vee^{\mathrm{r}} \exists^{\mathrm{r}}_{y}\exists^{\mathrm{d}}_{d}(({}^{\mathrm{co}}Iy \vee^{\mathrm{d}} Py) \wedge^{\mathrm{l}} f(x) \operatorname{eqd} \mathrm{Av}_d(y)))).$$

Let f be given and assume $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}\, f$. By using Read^{-} for $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}\, f$, it suffices to prove the following two step cases

$$\forall^{\mathrm{c}}_{e}\forall^{\mathrm{nc}}_{f}(f[\mathbb{I}] \subseteq \mathbb{I}_e \to {}^{\mathrm{co}}\mathrm{Write}(\mathrm{Out}_e \circ f) \to Qf)$$

and

$$\begin{aligned}\forall^{\mathrm{nc}}_{f}(&\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}(f \circ \mathrm{In}_{\mathsf{L}}) \to Q(f \circ \mathrm{In}_{\mathsf{L}}) \to\\ &\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}(f \circ \mathrm{In}_{\mathsf{M}}) \to Q(f \circ \mathrm{In}_{\mathsf{M}}) \to\\ &\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}(f \circ \mathrm{In}_{\mathsf{R}}) \to Q(f \circ \mathrm{In}_{\mathsf{R}}) \to Qf),\end{aligned}$$

where $Q := \{f \mid \forall^{\mathrm{nc}}_{x}({}^{\mathrm{co}}Ix \to f(x) \operatorname{eqd} \mathrm{Z} \vee^{\mathrm{r}} \exists^{\mathrm{r}}_{y}\exists^{\mathrm{d}}_{d}(({}^{\mathrm{co}}Iy \vee^{\mathrm{d}} Py) \wedge^{\mathrm{l}} f(x) \operatorname{eqd} \mathrm{Av}_d(y)))\}$. *First step case.* Let e and f be given and assume $f[\mathbb{I}] \subseteq \mathbb{I}_e$ and ${}^{\mathrm{co}}\mathrm{Write}(\mathrm{Out}_e \circ f)$. Also let x be given and assume ${}^{\mathrm{co}}Ix$. We prove the right disjunct of the conclusion. Use $(\exists^{\mathrm{r}})^{+}$ and $(\exists^{\mathrm{d}})^{+}$ with $y = (\mathrm{Out}_e \circ f)(x)$ and $d = e$, respectively. The equality is obvious from (VaOut) and (AvVaIdent). In order to prove Py, we use $(\exists^{\mathrm{r}})^{+}$ with $\mathrm{Out}_e \circ f$ and with x, then three conjuncts are trivial. *Second step case.* Let f be given and assume $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}(f \circ \mathrm{In}_e)$ and $\forall^{\mathrm{nc}}_{x}({}^{\mathrm{co}}Ix \to (f \circ \mathrm{In}_e)(x) \operatorname{eqd} \mathrm{Z} \vee^{\mathrm{r}} \exists^{\mathrm{r}}_{y}\exists^{\mathrm{d}}_{d}(({}^{\mathrm{co}}Iy \vee^{\mathrm{d}} Py) \wedge^{\mathrm{l}} (f \circ$

$\mathrm{In}_e)(x)\,\mathrm{eqd}\,\mathrm{Av}_d(y)))$ for each $e \in \mathbf{SD}$. Also let x be given and assume ${}^{\mathrm{co}}Ix$. We prove $f(x)\,\mathrm{eqd}\,\mathrm{Z} \vee^{\mathrm{r}} \exists^{\mathrm{r}}_y \exists^{\mathrm{d}}_d(({}^{\mathrm{co}}Iy \vee^{\mathrm{d}} Py) \wedge^{\mathrm{l}} f(x)\,\mathrm{eqd}\,\mathrm{Av}_d(y))$. Applying ${}^{\mathrm{co}}I^+$ to ${}^{\mathrm{co}}Ix$, there are two cases from $x\,\mathrm{eqd}\,\mathrm{Z} \vee^{\mathrm{r}} \exists^{\mathrm{r}}_{y_0} \exists^{\mathrm{d}}_{d_0}({}^{\mathrm{co}}Iy_0 \wedge^{\mathrm{l}} x\,\mathrm{eqd}\,\mathrm{Av}_{d_0}(y_0))$. *Left case.* Assume $x\,\mathrm{eqd}\,\mathrm{Z}$. Applying the induction hypothesis for $e = \mathsf{M}$ to ${}^{\mathrm{co}}I(\mathrm{Z})$, $(f \circ \mathrm{In}_{\mathsf{M}})(\mathrm{Z})\,\mathrm{eqd}\,\mathrm{Z} \vee^{\mathrm{r}} \exists^{\mathrm{r}}_y \exists^{\mathrm{d}}_e(({}^{\mathrm{co}}Iy \vee^{\mathrm{d}} Py) \wedge^{\mathrm{l}} (f \circ \mathrm{In}_{\mathsf{M}})(\mathrm{Z})\,\mathrm{eqd}\,\mathrm{Av}_e(y))$ holds, and moreover $(f \circ \mathrm{In}_{\mathsf{M}})(\mathrm{Z})\,\mathrm{eqd}\,f(\mathrm{Z})$ holds by means of (AvIn) and (AvZero), hence the goal is implied. *Right case.* There exist y_0 and d_0 such that ${}^{\mathrm{co}}Iy_0$ and $x\,\mathrm{eqd}\,\mathrm{Av}_{d_0}(y_0)$ hold. Applying the induction hypothesis corresponding to d_0 to ${}^{\mathrm{co}}Iy_0$, $(f \circ \mathrm{In}_{d_0})(y_0)\,\mathrm{eqd}\,\mathrm{Z} \vee^{\mathrm{r}} \exists^{\mathrm{r}}_y \exists^{\mathrm{d}}_e(({}^{\mathrm{co}}Iy \vee^{\mathrm{d}} Py) \wedge^{\mathrm{l}} (f \circ \mathrm{In}_{d_0})(y_0)\,\mathrm{eqd}\,\mathrm{Av}_e(y))$ holds and we are done because $(f \circ \mathrm{In}_{d_0})(y_0)\,\mathrm{eqd}\,f(x)$ holds by the assumption $x\,\mathrm{eqd}\,\mathrm{Av}_{d_0}(y_0)$ and (AvIn). □

3.6.3 Program Extraction

We extract a program from the proof of 3.6.4. Note that Remark 3.5.15 applies in this section as well. Suppose that u is a variable name ranging over $\mathbf{W} \times \mathbf{I}$ and q ranging over $\mathbf{SD} \times \mathbf{I}$. The result is app which is shown in Figure 3.20. Intuitively, this function computes

$$
\begin{array}{l}
\lambda_{w,ds}({}^{\mathrm{co}}\mathcal{R}^{\mathbf{W}\times\mathbf{I}}_{\mathbf{I}}\langle w, ds\rangle \\
\qquad \lambda_u(\mathrm{Case}\ \mathcal{D}_{\mathbf{W}}(\pi_0 u)\ \mathrm{of} \\
\qquad\qquad \lambda\_\mathrm{Case}\ \mathcal{D}(\pi_1 u)\ \mathrm{of} \\
\qquad\qquad\qquad \lambda\_\mathsf{InL}\,\mathsf{U} \\
\qquad\qquad\qquad \lambda_q \mathsf{InR}\langle \pi_0 q, \mathsf{InL}(\pi_1 q)\rangle \\
\qquad\qquad \lambda_r(\mathrm{Case}\ \mathtt{cApplyAux}(r(\pi_1 u))\ \mathrm{of}\ \lambda\_(\mathsf{InL}\,\mathsf{U})\,\lambda_v(\mathsf{InR}\,v))))
\end{array}
$$

Figure 3.20: $\mathsf{app}^{\mathbf{W}\to\mathbf{I}\to\mathbf{I}}$

a trace from the root of the given type-0 function to the above by choosing at Get a branch indicated by the input type-0 real number. The output is an SDS which consists of signed digits which occur at Put on the computed trace.

Assume w and ds of types $\mathbf{W}$ and $\mathbf{I}$, respectively, are given. The costep term first destruct w. If $\mathcal{D}w$ is $\mathsf{InL}\,\mathsf{U}$, the output is identical to the given ds. Otherwise, it results in an $\mathbf{R_W}$-total ideal r. We apply $\mathtt{cApplyAux}$ to this r and ds, then it determines that the result SDS is either the empty list or a pair of the head and the tail, i.e., a signed digit and an SDS, respectively.

We extract from the Lemma 3.6.5 a program $\mathtt{cApplyAux}$ shown in Figure 3.21. Let f be a variable name ranging over $\mathbf{I} \to \mathbf{U} + (\mathbf{SD} \times (\mathbf{I} + \mathbf{W} \times \mathbf{I}))$. For the input r and ds, this program computes by reading finite signed digits from ds a finite path in r, so that a signed digit is determined. There are two cases due to the recursion on r. If r is $\mathsf{Put}\,d\,w$ it does not touch ds and outputs $\mathsf{InR}\langle d, \mathsf{InR}\langle w, ds\rangle\rangle$. It makes the corecursion operator in app to produce a stream with the head d and the rest $\mathsf{app}\,w\,ds$. Otherwise, r is $\mathsf{Get}\,r_{\mathsf{L}}\,r_{\mathsf{M}}\,r_{\mathsf{R}}$. If $\mathcal{D}ds$ is $\mathsf{InL}\,\mathsf{U}$, it interprets the ds as a rational number $\mathsf{M}::\mathsf{M}::\ldots$ and the middle branch is

$$
\begin{array}{l}
\lambda_r(\mathcal{R}^{\mathbf{I}\to\mathbf{U}+(\mathbf{SD}\times(\mathbf{I}+\mathbf{W}\times\mathbf{I}))}_{\mathbf{R_W}} \\
\quad r \\
\quad \lambda_{d,w,ds}(\mathsf{InR}\langle d, \mathsf{InR}\langle w, ds\rangle\rangle) \\
\quad \lambda_{\_,f_\mathsf{L},\_,f_\mathsf{M},\_,f_\mathsf{R},ds}(\text{Case } \mathcal{D}_\mathbf{I} ds \text{ of} \\
\qquad\qquad\qquad \lambda_\_(f_\mathsf{M} ds) \\
\qquad\qquad\qquad \lambda_q(\text{Case } \pi_0 q \text{ of } \lambda_\_(f_\mathsf{L}(\pi_1 q))\, \lambda_\_(f_\mathsf{M}(\pi_1 q))\, \lambda_\_(f_\mathsf{R}(\pi_1 q)))))
\end{array}
$$

Figure 3.21: $\mathtt{cApplyAux}^{\mathbf{R_W}\to\mathbf{I}\to\mathbf{U}+(\mathbf{SD}\times(\mathbf{I}+\mathbf{W}\times\mathbf{I}))}$

taken to reach the leaf. Otherwise if $\mathcal{D}ds$ is $\mathsf{InR}\langle d, ds'\rangle$, it takes a branch corresponding to d and then the recursion carries on with ds'.

3.6.4 Experiments

Define a function $f(x) = 2x^2 - 1$ and consider its roots $\pm\frac{1}{\sqrt{2}}$. It is expected that the value $f(\pm\frac{1}{\sqrt{2}})$ is arbitrarily close to 0. We use `app` to compute the value of $f(\frac{1}{\sqrt{2}})$.

A type-1 uniformly continuous function for f is defined to be $\langle h, \alpha, \omega\rangle$ where $h\, a\, n = 2a^2 - 1$, $\alpha\, n = n + 1$ and $\omega\, n = n + 1$. Then by means of `ucf1to0`, we obtain the type-0 representation of $\langle h, \alpha, \omega\rangle$. On the other hand, the type-0 representation of $\frac{1}{\sqrt{2}}$ is computed by `cauchysds` and sqrt as in Section 3.3.4. Using `app`, we can compute $\mathtt{app}(\mathtt{ucf1to0}\, f)(\mathtt{cauchysds}(\mathrm{sqrt}\frac{1}{2}))$ in an SDS as follows:

$$\mathsf{M} :: \mathsf{M} :: \mathsf{M} :: \mathsf{M} :: \mathsf{M} :: \mathsf{M} :: \mathsf{M} :: \ldots.$$

3.7 Composition of Uniformly Continuous Functions

From formalized proofs we extract a program which computes the composition of two type-0 uniformly continuous functions. The input of the composition algorithm is two type-0 uniformly continuous functions and the output is a type-0 uniformly continuous function. For simplicity we regard a uniformly continuous function as a tree with 3-branching nodes and 1-branching nodes labelled with a signed digit. It takes a look at the root of the two input trees. When both of them are a 3-branching node, let the three subtrees of the first input be r_0, r_1 and r_2, then it makes a 3-branching node for the output with three subtrees computed by composing r_0, r_1 and r_2 with the second input, respectively. When the first input is a 3-branching node and the second one is a 1-branching node with a signed digit d, the output is a composition of the subtree of the first input corresponding to d, namely, the left, the middle, or the right, and the subtree of the second input. When the first input

is a 1-branching tree, it puts the very same 1-branching tree for the output and the rest is a composition of the subtree of the first input and the original second input.

3.7.1 Definitions

In order to talk about the composition of abstract functions, define a constant Cmp whose properties are given by the abstract theory.

Definition 3.7.1 (Cmp)**.** Cmp is a constant of type $\phi \to \phi \to \phi$. Cmp f g is abbreviated as $f \circ g$.

We extend the abstract theory of uniformly continuous functions in Section 3.5.1 with the following axioms.

Axiom 3.7.2.

$$\forall_f(f \circ \mathrm{Id}\,\mathrm{eqd}\,f) \qquad (\textsc{CompIdR})$$
$$\forall_f(\mathrm{Id} \circ f\,\mathrm{eqd}\,f) \qquad (\textsc{CompIdL})$$
$$\forall_d((\mathrm{Id} \circ \mathrm{In}_d)[\mathbb{I}] \subseteq \mathbb{I}_d) \qquad (\textsc{IdIn})$$
$$\forall_{f,d}(\mathrm{Out}_d \circ (\mathrm{Id} \circ \mathrm{In}_d)\,\mathrm{eqd}\,\mathrm{Id}) \qquad (\textsc{OutIdIn})$$
$$\forall_{f_1,f_2,d}(f_1[\mathbb{I}] \subseteq \mathbb{I}_d \to (f_1 \circ f_2)[\mathbb{I}] \subseteq \mathbb{I}_d) \qquad (\textsc{CompBound})$$
$$\forall_{f,p,ld}(f[\mathbb{I}] \subseteq \mathbb{I}_{p,l} \to (f \circ \mathrm{In}_d)[\mathbb{I}] \subseteq \mathbb{I}_{p,l}) \qquad (\textsc{UcfInputIn})$$
$$\forall_{f_1,f_2,d}(f_2[\mathbb{I}] \subseteq \mathbb{I}_d \to f_1 \circ f_2\,\mathrm{eqd}\,(f_1 \circ \mathrm{In}_d) \circ (\mathrm{Out}_d \circ f_2)) \qquad (\textsc{InOutIdent})$$
$$\forall_{f,d_1,d_2}(\mathrm{Out}_{d_1} \circ (f \circ \mathrm{In}_{d_2})\,\mathrm{eqd}\,(\mathrm{Out}_{d_1} \circ f) \circ \mathrm{In}_{d_2}) \qquad (\textsc{AssocOutIn})$$
$$\forall_{f_1,f_2,d}((f_1 \circ f_2) \circ \mathrm{In}_d\,\mathrm{eqd}\,f_1 \circ (f_2 \circ \mathrm{In}_d)) \qquad (\textsc{AssocCompIn})$$
$$\forall_{f_1,f_2,d}(\mathrm{Out}_d \circ (f_1 \circ f_2)\,\mathrm{eqd}\,(\mathrm{Out}_d \circ f_1) \circ f_2) \qquad (\textsc{AssocOutComp})$$

We also use the predicates ${}^{\mathrm{co}}I$, Read and ${}^{\mathrm{co}}$Write from Definition 3.3.6, 3.5.6 and 3.5.7, respectively.

3.7.2 Proofs

We extract from a proof a program which composes two type-0 uniformly continuous functions. Let variable names f, g and h range over ϕ.

Proposition 3.7.3. $\forall^{\mathrm{nc}}_{f,g}({}^{\mathrm{co}}\mathrm{Write}\,f \to {}^{\mathrm{co}}\mathrm{Write}\,g \to {}^{\mathrm{co}}\mathrm{Write}(f \circ g))$.

Proof. Let f and g be given and assume ${}^{\mathrm{co}}\mathrm{Write}\,f$ and ${}^{\mathrm{co}}\mathrm{Write}\,g$. We use ${}^{\mathrm{co}}\mathrm{Write}^+$ with a competitor predicate $P := \{h \mid \exists^{\mathrm{r}}_{f,g}({}^{\mathrm{co}}\mathrm{Write}\,f \wedge {}^{\mathrm{co}}\mathrm{Write}\,g \wedge f \circ g\,\mathrm{eqd}\,h)\}$. It suffices to show $\forall^{\mathrm{nc}}_h(Ph \to h\,\mathrm{eqd}\,\mathrm{Id} \vee^{\mathrm{r}} \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee^{\mathrm{d}} P}\,h)$. Let h be given and assume Ph, i.e., ${}^{\mathrm{co}}\mathrm{Write}\,f$, ${}^{\mathrm{co}}\mathrm{Write}\,g$ and $f \circ g\,\mathrm{eqd}\,h$. We prove the goal $f \circ g\,\mathrm{eqd}\,\mathrm{Id} \vee^{\mathrm{r}} \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write} \vee^{\mathrm{d}} P}(f \circ g)$. Applying ${}^{\mathrm{co}}\mathrm{Write}^-$ to ${}^{\mathrm{co}}\mathrm{Write}\,f$ and to ${}^{\mathrm{co}}\mathrm{Write}\,g$, there are four cases from the following two disjunctions.

$$f\,\mathrm{eqd}\,\mathrm{Id} \vee^{\mathrm{r}} \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}\,f, \quad g\,\mathrm{eqd}\,\mathrm{Id} \vee^{\mathrm{r}} \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}\,g.$$

(1) Assume f eqd Id and g eqd Id. The left disjunct of the goal holds due to (CompIdL) or (CompIdR). (2) Assume $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}\, f$ and g eqd Id. We prove the right disjunct of the goal. By (CompIdR), the goal is $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee^{\mathrm{d}}P}\, f$. By Read^{-} for $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}\, f$, it suffices to prove the following two step formulas.

$$\begin{aligned}
&\forall_f \forall^{\mathrm{c}}_d (f[\mathbb{I}] \subseteq \mathbb{I}_d \to {}^{\mathrm{co}}\mathrm{Write}(\mathrm{Out}_d \circ f) \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee^{\mathrm{d}}P}\, f),\\
&\forall_f (\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}(f \circ \mathrm{In}_{\mathsf{L}}) \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee^{\mathrm{d}}P}(f \circ \mathrm{In}_{\mathsf{L}}) \to\\
&\quad \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}(f \circ \mathrm{In}_{\mathsf{M}}) \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee^{\mathrm{d}}P}(f \circ \mathrm{In}_{\mathsf{M}}) \to\\
&\quad \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}(f \circ \mathrm{In}_{\mathsf{R}}) \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee^{\mathrm{d}}P}(f \circ \mathrm{In}_{\mathsf{R}}) \to\\
&\quad \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee^{\mathrm{d}}P}\, f).
\end{aligned}$$

For the first one, let f, d be given and assume $f[\mathbb{I}] \subseteq \mathbb{I}_d$ and ${}^{\mathrm{co}}\mathrm{Write}(\mathrm{Out}_d \circ f)$. Prove the goal $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee P}\, f$ by using Read^{+}_0 with d. It suffices to prove the premises $f[\mathbb{I}] \subseteq \mathbb{I}_d$ and $({}^{\mathrm{co}}\mathrm{Write} \vee^{\mathrm{d}} P)(\mathrm{Out}_d \circ f)$. The former one is in the assumption and the latter comes from the assumption by $(\vee^{\mathrm{d}})^{+}_0$. For the second one, let f be given and assume the premises. Since $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee P}(f \circ \mathrm{In}_d)$ holds for each $d \in \mathbf{SD}$, we apply Read^{+}_1 to prove $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee P}\, f$. (3) Similar to (2). (4) Assume $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}\, f$ and $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}\, g$. We prove $\forall^{\mathrm{nc}}_g({}^{\mathrm{co}}\mathrm{Write}\, g \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee^{\mathrm{d}}P}(f \circ g))$ by Read^{-}, in order the right disjunct of the goal holds. *Base case.* We prove

$$\forall^{\mathrm{nc}}_f \forall^{\mathrm{c}}_d (f[\mathbb{I}] \subseteq \mathbb{I}_d \to {}^{\mathrm{co}}\mathrm{Write}(\mathrm{Out}_d \circ f) \to \forall^{\mathrm{nc}}_g({}^{\mathrm{co}}\mathrm{Write}\, g \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee^{\mathrm{d}}P}(f \circ g))).$$

Let f and d be given and assume $f[\mathbb{I}] \subseteq \mathbb{I}_d$ and ${}^{\mathrm{co}}\mathrm{Write}(\mathrm{Out}_d \circ f)$. Furthermore, let g be given and assume ${}^{\mathrm{co}}\mathrm{Write}\, g$. We prove $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee^{\mathrm{d}}P}(f \circ g)$. Using Read^{+}_0 with d, it suffices to prove $(f \circ g)[\mathbb{I}] \subseteq \mathbb{I}_d$ and $({}^{\mathrm{co}}\mathrm{Write} \vee^{\mathrm{d}} P)(\mathrm{Out}_d \circ (f \circ g))$. The former one comes by (CompBound). We prove $P(\mathrm{Out}_d \circ (f \circ g))$, the right disjunct of the latter one. By (AssocOutComp), the conclusion comes from ${}^{\mathrm{co}}\mathrm{Write}(\mathrm{Out}_d \circ f)$ and ${}^{\mathrm{co}}\mathrm{Write}\, g$. *Step case.* We prove

$$\begin{aligned}
&\forall^{\mathrm{nc}}_f (\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}(f \circ \mathrm{In}_{\mathsf{L}}) \to \forall^{\mathrm{nc}}_g({}^{\mathrm{co}}\mathrm{Write}\, g \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee P}((f \circ \mathrm{In}_{\mathsf{L}}) \circ g)) \to\\
&\quad \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}(f \circ \mathrm{In}_{\mathsf{M}}) \to \forall^{\mathrm{nc}}_g({}^{\mathrm{co}}\mathrm{Write}\, g \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee P}((f \circ \mathrm{In}_{\mathsf{M}}) \circ g)) \to\\
&\quad \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}(f \circ \mathrm{In}_{\mathsf{R}}) \to \forall^{\mathrm{nc}}_g({}^{\mathrm{co}}\mathrm{Write}\, g \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee P}((f \circ \mathrm{In}_{\mathsf{R}}) \circ g)) \to\\
&\quad \forall^{\mathrm{nc}}_g({}^{\mathrm{co}}\mathrm{Write}\, g \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee P}(f \circ g))).
\end{aligned}$$

Let f be given and assume the premises. Also let g be given and assume ${}^{\mathrm{co}}\mathrm{Write}\, g$. Applying ${}^{\mathrm{co}}\mathrm{Write}^{-}$ to ${}^{\mathrm{co}}\mathrm{Write}\, g$, g eqd Id $\vee^{\mathrm{r}}$ $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}} g$ holds. From it, there are two cases. *Left case.* Assume g eqd Id, then $f \circ g$ eqd f by (CompIdR). Applying each induction hypothesis $\forall^{\mathrm{nc}}_g({}^{\mathrm{co}}\mathrm{Write}\, g \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee^{\mathrm{d}}P}((f \circ \mathrm{In}_d) \circ g))$ to Id and ${}^{\mathrm{co}}\mathrm{Write}\,\mathrm{Id}$, (CompIdR) and Read^{+}_1 imply the goal. *Right case.* Assume $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}\, g$. We prove $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee P}(f \circ g)$ by using Read^{-} for $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}\, g$. *Side base case.* We prove

$$\forall^{\mathrm{nc}}_g \forall^{\mathrm{c}}_d (g[\mathbb{I}] \subseteq \mathbb{I}_d \to {}^{\mathrm{co}}\mathrm{Write}(\mathrm{Out}_d \circ g) \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee P}(f \circ g)).$$

Let g and d be given and assume $g[\mathbb{I}] \subseteq \mathbb{I}_d$ and ${}^{\mathrm{co}}\mathrm{Write}(\mathrm{Out}_d \circ g)$. Applying the assumption $\forall_g^{\mathrm{nc}}({}^{\mathrm{co}}\mathrm{Write}\, g \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee^{\mathrm{d}} P}((f \circ \mathrm{In}_d) \circ g))$ to ${}^{\mathrm{co}}\mathrm{Write}(\mathrm{Out}_d \circ g)$ and using (InOutIdent), we can prove $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee^{\mathrm{d}} P}(f \circ g)$. *Side step case.* We prove

$$\begin{aligned}\forall_g^{\mathrm{nc}}(&\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}(g \circ \mathrm{In}_{\mathsf{L}}) \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee P}(f \circ (g \circ \mathrm{In}_{\mathsf{L}})) \to \\ &\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}(g \circ \mathrm{In}_{\mathsf{M}}) \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee P}(f \circ (g \circ \mathrm{In}_{\mathsf{M}})) \to \\ &\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}(g \circ \mathrm{In}_{\mathsf{R}}) \to \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee P}(f \circ (g \circ \mathrm{In}_{\mathsf{R}})) \to \\ &\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee P}(f \circ g)).\end{aligned}$$

Let g be given and assume the premises, then we can prove $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}\vee P}(f \circ g)$ by means of Read_1^+. □

Remark 3.7.4. Berger's proof of the proposition of composition (Proposition 4.2 in [Ber11]) uses a lemma $\forall_f^{\mathrm{nc}}({}^{\mathrm{co}}\mathrm{Write}\, f \to {}^{\mathrm{co}}\mathrm{Write}(f \circ \mathrm{In}_{i,d}))$ (Lemma 4.1 in [Ber11]). This is needed to deal with arbitrary n-ary functions. Since we restricted our case study to unary functions, it is unnecessary to consider this lemma.

3.7.3 Program Extraction

We extract from the proof of Proposition 3.7.3 a program `cmp` as shown in Figure 3.22. Inside of `cmp`, there is an occurrence of `M` which stands for a program separately shown in Figure 3.23. Suppose that w_0 and w_1 of type $\mathbf{W}$ are given to `cmp`. Unfolding the corecursion

$$\begin{aligned}&\lambda_{w_0,w_1}({}^{\mathrm{co}}\mathcal{R}^{\mathbf{W}\times\mathbf{W}}_{\mathbf{W}} \\ &\quad \langle w_0, w_1\rangle \\ &\quad \lambda_{uw}(\mathrm{Case}\ \mathcal{D}(\mathrm{Case}\ uw\ \mathrm{of}\ \lambda_{w_2,w_3} w_2)\ \mathrm{of} \\ &\qquad \lambda\_\,\mathrm{Case}\ \mathcal{D}(\mathrm{Case}\ uw\ \mathrm{of}\ \lambda_{w_2,w_3} w_3)\ \mathrm{of} \\ &\qquad\quad \lambda\_(\mathsf{InL}\,\mathsf{U}) \\ &\qquad\quad \lambda_{rw}(\mathsf{InR}(\mathcal{R}_{\mathbf{R_W}}\, rw \\ &\qquad\qquad\qquad \lambda_{d,w_4}(\mathsf{Put}\, d\, (\mathsf{InL}\, w_4)) \\ &\qquad\qquad\qquad \lambda_{\_,q_0,\_,q_1,\_,q_2}(\mathsf{Get}\, q_0\, q_1\, q_2))) \\ &\qquad \lambda_{rw}(\mathrm{Case}\ \mathcal{D}(\mathrm{Case}\ uw\ \mathrm{of}\ \lambda_{w_2,w_3} w_3)\ \mathrm{of} \\ &\qquad\quad \lambda\_(\mathsf{InR}(\mathcal{R}_{\mathbf{R_W}}\, rw \\ &\qquad\qquad\qquad \lambda_{d,w_4}(\mathsf{Put}\, d\, (\mathsf{InL}\, w_4)) \\ &\qquad\qquad\qquad \lambda_{\_,q_0,\_,q_1,\_,q_2}(\mathsf{Get}\, q_0\, q_1\, q_2))) \\ &\qquad\quad \lambda\_(\mathtt{M}\, rw\, (\mathrm{Case}\ ww\ \mathrm{of}\ \lambda_{w_0,w_1} w_1)))))\end{aligned}$$

Figure 3.22: $\mathtt{cmp}^{\mathbf{W}\to\mathbf{W}\to\mathbf{W}}$

operator, there are four cases comming from $\mathcal{D}w_0$ and $\mathcal{D}w_1$ as in the proof. If both of them are the left injection of the unit, it results in Stop. If one is the right injection of some r of type $\mathbf{R_W}$ and the other is the left injection of the unit, it works so that the corecursion produces $\mathsf{Cont}\, r$. By means of the corecursion on $\mathbf{W}$, every $\mathbf{W}$-cototal ideals appearing at the end of r is tagged as the next output by being injected to the left at Put. Considering the type of the costep term $\tau \to \mathbf{U} + \mathbf{R}_{\mathbf{W}+\tau}$, the value is injected at the position of $\mathbf{W}$ by InL. The last case is taken care of by $\mathtt{M}$. If both of the results of destructing w_0 and w_1 are the right injection, suppose that $\mathcal{D}w_0 = \mathsf{InR}\, r$. Inside of the recursion on this r, it generates

$$
\begin{array}{l}
\lambda_{rw,w}(\mathsf{InR}(\mathcal{R}_{\mathbf{R_W}}\ rw \\
\qquad \lambda_{d,w_0,w_1}(\mathsf{Put}\, d\, (\mathsf{InR}\langle w_0, w_1\rangle)) \\
\qquad \lambda_{\_,f_0,\_,f_1,\_,f_2,w_0}(\text{Case } \mathcal{D}w_0 \text{ of} \\
\qquad\qquad \lambda_{\_}(\mathsf{Get}(f_0 w_0)(f_1 w_0)(f_2 w_0)) \\
\qquad\qquad \lambda_{rw}(\mathcal{R}\ rw \\
\qquad\qquad\qquad \lambda_{d,w_1}(\text{Case } d \text{ of } (f_0 w_1) \\
\qquad\qquad\qquad\qquad\qquad\qquad (f_1 w_1) \\
\qquad\qquad\qquad\qquad\qquad\qquad (f_2 w_1)) \\
\qquad\qquad\qquad \lambda_{\_,r_0,\_,r_1,\_,r_2}(\mathsf{Get}\, r_0\, r_1\, r_2))) \\
\qquad w))
\end{array}
$$

Figure 3.23: $\mathtt{M}^{\mathbf{R_W}\to\mathbf{W}\to\mathbf{U}+\mathbf{R}_{\mathbf{W}+\mathbf{W}\times\mathbf{W}}}$

one $\mathbf{R_W}$-total ideal by reading off finite information of w_1 from its root. The number of $\mathbf{R_W}$-total trees to read depends on the height of r.

3.7.4 Experiments

We compose two uniformly continuous functions $f_1(x) = 2x^2 - 1$ and $f_2(x) = \sqrt{\frac{\frac{x}{2}+1}{2}}$. We define type-1 uniformly continuous functions $\langle h_1, \alpha_1, \omega_1\rangle$ and $\langle h_2, \alpha_2, \omega_2\rangle$ as follows.

$$
\begin{array}{l}
h_1\, a\, n = a^2 - 1, \quad \alpha_1\, n = n + 1, \quad \omega_1\, n = n + 1, \\
h_2\, a\, n = \mathcal{R}\, n\, 1\, \lambda_{b,n}(\frac{b + \frac{c_a}{b}}{2}), \quad \alpha_2\, n = n + 1, \quad \omega_2\, n = n,
\end{array}
$$

where $c_a = \frac{\frac{a}{2}+1}{2}$. The type-0 represented functions are obtained by means of `ucf1to0`. Then `cmp` computes the composition, which results in $g(x) = \frac{x}{2}$.

3.8 Integration

For the last case study of this chapter, we study definite integration. We extract a program to compute the definite integral of a type-0 uniformly continuous function in type-1 real numbers. As in the last sections, we consider uniformly continuous functions with domain and codomain being $[-1, 1]$. For simplicity of the formalization, we consider the half of the integral value, so that the integration of a uniformly continuous function is in $[-1, 1]$ without loss of the information. The basic idea of integration is said to be "split and accumulate". The computational content of our proof is exactly an implementation of this idea.

3.8.1 Definitions

We define the abstract theory to study definite integral. Let ρ and ϕ be abstract types of real numbers and uniformly continuous functions, respectively. Suppose that x and y range over ρ, f ranges over ϕ, p and q range over $\mathbf{Q}$, d ranges over $\mathbf{SD}$ and n ranges over $\mathbf{N}$. As in Definition 3.3.2, Z is a constant of type ρ. We extend Axiom 3.3.5 by adding the followings.

Axiom 3.8.1 (Abstract theory of real numbers for definite integration)**.**

$$\forall_n(\mathrm{Z} \in \mathbb{I}_{0,n}) \tag{RealZero}$$

$$\forall_{x,y,p,q,n}\left(x \in \mathbb{I}_{p,n} \to y \in \mathbb{I}_{q,n} \to \frac{x+y}{2} \in \mathbb{I}_{\frac{p+q}{2},n}\right) \tag{RealAvrg}$$

The rest is to deal with integration in the abstract setting.

Definition 3.8.2 (Integration)**.** Let $\int^{\mathrm{H}}$ be a constant of type $\phi \to \rho$.

This constant is intended to mean the half of the definite integration. The properties of this integration is specified by the following axioms. The constants Id, Out_d and In_d are as defined in Section 3.5.

Axiom 3.8.3 (Abstract Theory of Integration)**.**

$$\forall_f\left(\int^{\mathrm{H}} f \in \mathbb{I}\right) \tag{HIntBound}$$

$$\int^{\mathrm{H}} \mathrm{Id}\,\mathrm{eqd}\,\mathrm{Z} \tag{HIntId}$$

$$\forall_{f,d}\left(\int^{\mathrm{H}} f\,\mathrm{eqd}\,\frac{1}{2}\left(\int^{\mathrm{H}}(\mathrm{Out}_d \circ f) + d\right)\right) \tag{HIntOut}$$

$$\forall_f\left(\int^{\mathrm{H}} f\,\mathrm{eqd}\,\frac{1}{2}\left(\int^{\mathrm{H}}(f \circ \mathrm{In}_{\mathsf{L}}) + \int^{\mathrm{H}}(f \circ \mathrm{In}_{\mathsf{R}})\right)\right) \tag{HIntIn}$$

The definite integration is bounded in $[-1,1]$ by (HIntBound). The definite integral of the identity function is 0 due to (HIntId). Recall that intuitively $\mathrm{Out}_d \circ f(x) = 2f(x) - d$. A special case of the property $\int f(x)dx = \frac{1}{2}\int 2f(x)dx$ is given by (HIntOut). In usual settings, $\int_a^b f$ can be computed by $\int_a^c f + \int_c^b f$. A special case of this property holds by (HIntIn) which postulates that $\int_{-1}^1 f$ is equal to $\int_{-1}^0 f + \int_0^1 f$.

3.8.2 Proofs

We prove the following proposition from which we extract a program to compute integration.

Proposition 3.8.4.

$$\forall_f^{\mathrm{nc}}\left({}^{\mathrm{co}}\mathrm{Write}\, f \to \forall_n^{\mathrm{c}}\exists_p^{\mathrm{l}}\left(\int^{\mathrm{H}} f \in \mathbb{I}_{p,n}\right)\right).$$

Proof. The statement is same as $\forall_n^{\mathrm{c}}\forall_f^{\mathrm{nc}}({}^{\mathrm{co}}\mathrm{Write}\, f \to \exists_p^{\mathrm{l}}(\int^{\mathrm{H}} f \in \mathbb{I}_{p,n}))$. We prove it by induction on n. *Case* $n = 0$. Our goal is $\forall_f^{\mathrm{nc}}({}^{\mathrm{co}}\mathrm{Write}\, f \to \exists_p^{\mathrm{l}}(\int^{\mathrm{H}} f \in \mathbb{I}_{p,0}))$. Let f be given and assume ${}^{\mathrm{co}}\mathrm{Write}\, f$. Let p be 0, then it is clear by (HIntBound). *Case* $n \mapsto n+1$. From the following induction hypothesis,

$$\forall_f^{\mathrm{nc}}\left({}^{\mathrm{co}}\mathrm{Write}\, f \to \exists_p^{\mathrm{l}}\left(\int^{\mathrm{H}} f \in \mathbb{I}_{p,n}\right)\right), \tag{3.8}$$

we prove $\forall_f^{\mathrm{nc}}({}^{\mathrm{co}}\mathrm{Write}\, f \to \exists_p^{\mathrm{l}}(\int^{\mathrm{H}} f \in \mathbb{I}_{p,n+1}))$. Let f be given and assume ${}^{\mathrm{co}}\mathrm{Write}\, f$. We prove $\exists_p^{\mathrm{l}}(\int^{\mathrm{H}} f \in \mathbb{I}_{p,n+1})$. Applying ${}^{\mathrm{co}}\mathrm{Write}^+$ to ${}^{\mathrm{co}}\mathrm{Write}\, f$, there are two cases from $f \,\mathrm{eqd}\, \mathrm{Id} \vee^{\mathrm{r}} \mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}\, f$. *Left case.* Assume $f \,\mathrm{eqd}\, \mathrm{Id}$. Let p be 0, then our goal is $\int^{\mathrm{H}} \mathrm{Id} \in \mathbb{I}_{0,n+1}$, which holds by (RealZero). *Right case.* Assume $\mathrm{Read}_{{}^{\mathrm{co}}\mathrm{Write}}\, f$ and use Read^- for it. *Side base case.* We prove

$$\forall_f^{\mathrm{nc}}\forall_d^{\mathrm{c}}\left(f[\mathbb{I}] \subseteq \mathbb{I}_d \to {}^{\mathrm{co}}\mathrm{Write}(\mathrm{Out}_d \circ f) \to \exists_p^{\mathrm{l}}\left(\int^{\mathrm{H}} f \in \mathbb{I}_{p,n+1}\right)\right).$$

Let f, d be given and assume $f[\mathbb{I}] \subseteq \mathbb{I}_d$ and ${}^{\mathrm{co}}\mathrm{Write}(\mathrm{Out}_d \circ f)$. Our goal is $\exists_p^{\mathrm{l}}(\int^{\mathrm{H}} f \in \mathbb{I}_{p,n+1})$. Applying (3.8) to ${}^{\mathrm{co}}\mathrm{Write}(\mathrm{Out}_d \circ f)$, there is some p' such that $\int^{\mathrm{H}}(\mathrm{Out}_d \circ f) \in \mathbb{I}_{p',n}$, which implies $\frac{1}{2}(\int^{\mathrm{H}}(\mathrm{Out}_d \circ f) + d) \in \mathbb{I}_{\frac{p'+d}{2},n+1}$ by (AvIntro). It moreover implies $\int^{\mathrm{H}} f \in \mathbb{I}_{\frac{p'+d}{2},n+1}$

by (HIntOut). We use $(\exists^{\mathrm{l}})^+$ with $\frac{p'+d}{2}$ to derive the goal. *Side step case.* We prove

$$\forall_f^{\mathrm{nc}}\Bigg(\mathrm{Read}_{\mathrm{coWrite}}(f\circ \mathrm{In}_{\mathsf{L}}) \to \exists_p^{\mathrm{l}}\left(\int^{\mathrm{H}}(f\circ\mathrm{In}_{\mathsf{L}})\in \mathbb{I}_{p,n+1}\right)\to$$
$$\mathrm{Read}_{\mathrm{coWrite}}(f\circ \mathrm{In}_{\mathsf{M}}) \to \exists_p^{\mathrm{l}}\left(\int^{\mathrm{H}}(f\circ\mathrm{In}_{\mathsf{M}})\in \mathbb{I}_{p,n+1}\right)\to$$
$$\mathrm{Read}_{\mathrm{coWrite}}(f\circ \mathrm{In}_{\mathsf{R}}) \to \exists_p^{\mathrm{l}}\left(\int^{\mathrm{H}}(f\circ\mathrm{In}_{\mathsf{R}})\in \mathbb{I}_{p,n+1}\right)\to \exists_p^{\mathrm{l}}\left(\int^{\mathrm{H}} f\in \mathbb{I}_{p,n+1}\right)\Bigg).$$

Let f be given and assume the premises. To prove our goal $\exists_p^{\mathrm{l}}(\int^{\mathrm{H}} f\in\mathbb{I}_{p,n+1})$, the following two side induction hypotheses are used:

$$\exists_p^{\mathrm{l}}\left(\int^{\mathrm{H}}(f\circ\mathrm{In}_{\mathsf{L}})\in\mathbb{I}_{p,n+1}\right)\quad\text{and}\quad\exists_p^{\mathrm{l}}\left(\int^{\mathrm{H}}(f\circ\mathrm{In}_{\mathsf{R}})\in\mathbb{I}_{p,n+1}\right).$$

From the above side induction hypotheses, there are p_{L} and p_{R} such that $\int^{\mathrm{H}}(f\circ\mathrm{In}_{\mathsf{L}})\in\mathbb{I}_{p_{\mathsf{L}},n+1}$ and $\int^{\mathrm{H}}(f\circ\mathrm{In}_{\mathsf{R}})\in\mathbb{I}_{p_{\mathsf{R}},n+1}$. Using (RealAvrg) and (HIntIn), $\int^{\mathrm{H}} f\in\mathbb{I}_{\frac{p_{\mathsf{L}}+p_{\mathsf{R}}}{2},n+1}$ holds. Then the goal comes by $(\exists^{\mathrm{l}})^+$. □

3.8.3 Program Extraction

We extract from Proposition 3.8.2 the program `int` which computes the half of the integral of a type-0 uniformly continuous function. This program reads the given type-0 function

$$\begin{array}{l}\lambda_{w,n}(\mathcal{R}^{\mathbf{W}\to\mathbf{Q}}_{\mathbf{N}}\, n\,\lambda\_0\\ \qquad\lambda_{\_,g^{\mathbf{W}\to\mathbf{Q}},w_0}(\mathrm{Case}\ \mathcal{D}w_0\ \mathrm{of}\\ \qquad\qquad \mathsf{InL}\,\mathsf{U}\to 0\\ \qquad\qquad \mathsf{InR}\,r\to\mathcal{R}^{\mathbf{Q}}_{\mathbf{R_W}}\,r\,\lambda_{d,w_2}(\frac{g(w_2)+d}{2})\\ \qquad\qquad\qquad\lambda_{\_,p_{\mathsf{L}},\_,p_{\mathsf{M}},\_,p_{\mathsf{R}}}(\frac{p_{\mathsf{L}}+p_{\mathsf{R}}}{2}))\\ \qquad w)\end{array}$$

Figure 3.24: $\mathtt{int}^{\mathbf{W}\to\mathbf{N}\to\mathbf{Q}}$

to accumulate the possible output digits to compute the definite integral. The second argument is a number n to specify the bound of the computation in such a way that the program processes the read-write machine from its root up to the n-th $\mathbf{R_W}$-total ideals. At a branch, the recursively computed integral from the middle interval, which is denoted by

an unused variable p_{M}, is ignored because it suffices to see value on the left and the right subintervals, namely, $[-1, 0]$ and $[0, 1]$. At a leaf, the output digit is counted to contribute to the output with concerning its height in the tree. The base case of the the side recursion takes care of it by computing a value with division by 2. Consider $\mathbf{R_W}$-total ideals inside of a $\mathbf{W}$-cototal $\mathbf{R_W}$-total ideal. In the n-th height from the root, there are 2^n signed digits to count at the `Put` nodes. Indeed, this `int` is an implementation of what Bishop and Bridges [BB85] have mentioned, namely, *the averaging process*.

3.8.4 Experiments

We compute half of the definite integral for the function $f(x) \coloneqq \sqrt{x+2} - 1$. This integrand is defined to be $\langle h, \alpha, \omega \rangle$, where $h\, a\, n \coloneqq (\mathcal{R}_{\mathbf{N}}\, n\, 1\, \lambda_{\_,b}\, (\frac{b + \frac{a+2}{b}}{2})) - 1$, $\alpha\, n \coloneqq n+1$, and $\omega\, n \coloneqq n$ where $\mathrm{Pred} : \mathbf{N} \to \mathbf{N}$ is the predecessor function given in Example 2.2.38. We give two arguments to `int`, a type-0 function and a natural number. The first input to `int` is computed by `ucf1to0` from the above type-1 function. Specifying 8 as the second input, the output is $\frac{1633}{4096}$ whose decimal expansion is $0.398681640625\cdots$. Comparing our result with the manually calculated definite integral $\frac{1}{2}\int_{-1}^{1} f(x)dx = \sqrt{3} - \frac{4}{3}$, the error is $0.00003583\cdots$, which is smaller than $2^{-8} = 0.00390625$.

3.9 Notes

We describe related work, future work and remarks for concluding this chapter.

3.9.1 Exact Real Arithmetic via Streams

An early work on algorithms of stream represented real numbers is by Wiedmer [Wie77, Wie80]. He describes exact computation of real numbers and foundation of computability of infinite objects, and shows many case studies including the average of real numbers in the SDS representation. Ciaffaglione and Di Gianantonio [CG99, CG00, CG06] study the stream representation of real numbers by means of coinduction in Coq, namely, in a logical framework. Plume [Plu98] describes an informative report on exact real arithmetic and an implementation of a calculator for exact real number computation. His work includes an implementation of the average of signed digit streams as which our extracted program is essentially the same. Ghani, Hancock and Pattinson [GHP06, HPG09] study type-0 uniformly continuous functions, so-called stream processors, in coalgebraic setting. Abel [Abe07] studies mixed inductive/coinductive types and describes the composition of stream processors in a setting of equi-inductive types and equi-coinductive types. Formal certification of algorithms of exact real arithmetic is studied by Hou [Hou06], Berger and Hou [BH08]. They have certified algorithms which compute multiplication, division and limit of real sequences as well as translators between Cauchy reals and SDSs and average.

3.9.2 Program Extraction

Berger and Seisenberger [BS10, BS12] study a framework of program extraction supporting induction and coinduction, and informally reading-off the extracted program for the average of type-0 real numbers. Berger [Ber09, Ber11] gives a case study for type-0 representation of uniformly continuous functions, so-called read and write machines. Being heavily based on their work, we have given computer implemented case studies on exact real arithmetic. An extraction of the average program is done in Agda by Chuang [Chu11] by making use of Agda *postulates*. In contrast to the case of TCF (cf. also [MS13]), it requires a special care to guarantee that Agda *postulates* do not prevent normalization. It is taken care of by non-computational logical connectives in TCF and Minlog. Chuang [Chu11] moreover studies extraction of multiplication of exact real numbers in Agda.

3.9.3 Abstract Theory

The idea to make use of abstract theory in the context of program extraction is due to Berger [Ber09]. In our case studies, axioms are supposed to be valid in some model. For example, we can take a concrete theory of uniformly continuous functions in $[-1, 1]$, e.g. formulated by type-1 uniformly continuous functions. Instantiating the abstract type ϕ of uniformly continuous functions by the concrete type of type-1 uniformly continuous functions, the axioms can be proven. An issue arises when we consider to give arguments to an extracted program, although the extracted program itself is certified by soundness. A hand prepared input given to an extracted program should also be correct in the theory of program extraction. For example, a hand prepared type-1 uniformly continuous function $\langle \lambda_{a,n}(-a), n+1, \mathrm{Pred}\ n\rangle$ in Section 3.5.4 should be a realizer of $C\,f$ for some f. If we can specify a term f of the abstract type ϕ or instantiate the abstract theory by a concrete one, the problem is solved.

3.9.4 Formalizing n-Ary Uniformly Continuous Functions

According to Berger, n-ary type-0 uniformly continuous functions can be viewed as 3^n-branching trees [Ber09]. Although the theoretical treatment can be cleanly generalized through the general view by Berger, it still requires the base theory to be able to formalize algebras, predicates and abstract theories in a generalized way. We consider an extension of the algebra $\mathbf{R}_\alpha$ and $\mathbf{W}$. Let $\mathbf{SD}^n$ be an n-products of $\mathbf{SD}$. For instance, $\mathbf{SD}^0$ is empty, $\mathbf{SD}^1$ is same as $\mathbf{SD}$ and $\mathbf{SD}^2$ is a type of pairs of $\mathbf{SD}$. We define $\mathbf{R}^n_\alpha$ to be $\mu_\xi(\mathbf{SD} \to \alpha \to \xi, (\mathbf{SD}^n \to \xi) \to \xi)$. The algebra $\mathbf{W}^n$ is defined to be $\mu_\xi(\xi, \mathbf{R}^n_\xi \to \xi)$. Then, $\mathbf{W}^0$ is another way to represent $\mathbf{I}$, $\mathbf{W}^1$ is same as $\mathbf{W}$ and $\mathbf{W}^n$ consists of finite trees of $\mathbf{R}^n_{\mathbf{W}^n}$, where there are 3^n branches at the get node and the put node carries $\mathbf{SD}^n$ and $\mathbf{W}^n$. This represents n-ary uniformly continuous functions. In the same manner, one can also consider an extension of the predicates Read and $^{\mathrm{co}}$Write. Let f range over ϕ^n, an abstract n-ary uniformly continuous function, and e range over $\mathbf{SD}^n$. The constants Id^n, Out_n and In_n of types ϕ^n, $\mathbf{SD}^n \to \phi^n \to \phi^n$ and $\mathbf{SD}^n \to \phi^n \to \phi^n$, respectively, can be given as well.

We denote by $\mathbb{I}^n$ the unit interval of n-dimension. Then, the introduction axioms and the closure axiom look as follows. Let d and e range over $\mathbf{SD}$ and $\mathbf{SD}^n$, respectively.

$$\forall_f \forall^{\mathrm{c}}_d (f[\mathbb{I}^n] \subseteq \mathbb{I}_d \to X(\mathrm{Out}_{n,d} \circ f) \to \mathrm{Read}^n_X f), \qquad (\mathrm{Read}^n)^+_0$$
$$\forall_f (\forall^{\mathrm{c}}_e (\mathrm{Read}^n_X (f \circ \mathrm{In}_{n,e})) \to \mathrm{Read}^n_X f), \qquad (\mathrm{Read}^n)^+_1$$
$$\forall_f ({}^{\mathrm{co}}\mathrm{Write}^n f \to f \operatorname{eqd} \mathrm{Id}^n \vee \mathrm{Read}^n_{{}^{\mathrm{co}}\mathrm{Write}^n} f). \qquad ({}^{\mathrm{co}}\mathrm{Write}^n)^-$$

For $n = 0$, $(\mathrm{Read}^n)^+_0$ works as the right disjunct of the conclusion of ${}^{\mathrm{co}}I^-$, and $(\mathrm{Read}^n)^+_1$ becomes empty because the type of e is empty. Since they generally work for n-ary uniformly continuous functions, the application in Section 3.6 can be done by means of ${}^{\mathrm{co}}\mathrm{Write}^0$ and ${}^{\mathrm{co}}\mathrm{Write}^1$, the composition can be done by ${}^{\mathrm{co}}\mathrm{Write}^1$ and ${}^{\mathrm{co}}\mathrm{Write}^1$. Berger [Ber11] proves the following in his framework: Let f be an n-ary real function and g_i be m-ary ones for $i = 1, \ldots, n$, then ${}^{\mathrm{co}}\mathrm{Write}^n f$ and ${}^{\mathrm{co}}\mathrm{Write}^m g_i$ for each i imply ${}^{\mathrm{co}}\mathrm{Write}^m(f \circ (g_1, \ldots, g_n))$. It is a possible future work to extract from a formalized proof of the above proposition a program which computes a composition of n-ary and m-ary type-0 uniformly continuous functions.

Chapter 4

Concluding Remarks

4.1 Overview of this Thesis

In this Thesis we have studied program extraction from nested inductive and coinductive proofs. Our aim is to present a formalized theoretical framework for program extraction supporting nested inductive and coinductive definitions and a proof assistant within which program extraction can be practiced. We have enriched the Theory of Computable Functionals, so that nested inductive and coinductive definitions can be used for program extraction. We have implemented additional features for nested inductive and coinductive definitions to the proof assistant Minlog, which has been developed at the logic group of the University of Munich for more than 20 years. In order to practice program extraction in our framework with nestedness, we have formalized case studies in exact real arithmetic due to Berger and Seisenberger using Minlog. Non-trivial algorithms dealing with exact real numbers and uniformly continuous functions are mechanically extracted from proofs as executable programs in Minlog.

4.2 Contributions

We have enriched the Theory of Computable Functionals by introducing nestedness. Inductive and coinductive definitions can be combinated by nestedness in the proof theoretical side. The recursion and corecursion operators in the formal calculus T^+ have been enriched to support nestedness in the computational side. Concerning the realizability interpretation, the soundness theorem and the program extraction procedure have been extended to accommodate nestedness and coinductive definitions.

The theoretical improvement of the Theory of Computable Functionals has been implemented as additional features of the proof assistant Minlog, a computer implementation of the Theory of Computable Functionals. The current Minlog is able to deal with nested algebra definitions, constants, e.g. recursion and corecursion operators, on nested algebras, coinductive and nested definitions, proofs involving such definitions, and program extraction from nested inductive and coinductive proofs.

Within the Theory of Computable Functionals and the proof assistant Minlog, we formalized case studies due to Berger and Seisenberger in exact real arithmetic based on coinductive setting. Computational content from their proof, which was informally read off by Berger and Seisenberger, has been formally extracted by Minlog. Our formalization of exact real arithmetic in Minlog, which contains topics listed below, is currently a part of the downloadable Minlog package.

1. A translator from a rational number into a real number of type-0 [BMSS11].
2. A translator between a real number of type-1 and a real number of type-0 [Ber09].
3. The average of two real numbers of type-0 [Chu11, BS12, MS13].
4. A translator between a uniformly continuous function of type-1 and a uniformly continuous function of type-0 [Ber09, Ber11, MFS13].
5. The application of a uniformly continuous function of type-0 to a real number of type-0 [Ber09, Ber11].
6. The composition of two uniformly continuous functions of type-0 [Ber09, Ber11].
7. The definite integration of a uniformly continuous function of type-0 [Ber09, Ber11, MFS13].

4.3 Future Work

We illustrate three possible topics for future work. The first one is proof normalization involving coinductively defined predicates. The second is extending a feature of Minlog for automatic generation of program certifications involving nested definitions and coinductive definitions. The third is further exploration of program extraction in exact real arithmetic.

4.3.1 Proof Normalization

Through coinductive definitions, TCF is extended by additional closure and greatest-fixed-point axioms. As we saw, proof conversion rules are applicable if a greatest-fixed-point axiom is followed by a closure axiom, so that the use of closure axiom is eliminated. The proof normalization theorem for TCF with coinductive definitions is expected to hold, and it is a possible future work. Applying program extraction to a proof of the normalization theorem, an NbE algorithm supporting coinductive definitions can be obtained [Ber93, BBLS06].

4.3.2 Automatic Program Certification

The proof of Theorem 2.5.29 (Soundness) contains an informal algorithm to generate soundness proofs, namely, a given proof M of A can be transformed into a proof M' of $A^{\mathbf{r}}(\mathrm{et}(M))$. An implementation of the computational content of the proof of the soundness

theorem enables Minlog to automatically generate the proof M' from M. In other words, this is a feature to issue a certification of an extracted program. Currently, Minlog is able to deal with proofs involving simultaneous non-nested inductive definitions to generate soundness proofs. Minlog can automatically generate a soundness proof also for an extracted program from a nested and coinductive proof, provided the current feature is improved to cover the absent cases. This is a crucial step for Minlog to become a software development environment for certified programs.

The roadmap is as follows. The current Minlog feature for automation has to be extended to cover the cases of closure axioms and greatest-fixed-point axioms of coinductively defined predicates. The use of Lemma 2.5.30 (Soundness for monotonicity formulas) has to be taken care of. The computational content of this lemma generates a proof stating that a monotonicity formula is realized by the corresponding map operator. Generated soundness proofs are needed to complete the nested cases of I^- and ${}^{\mathrm{co}}I^+$ in the proof of Lemma 2.5.31 (Soundness for axioms of inductive and coinductive definitions).

4.3.3 Formalizing n-Ary Uniformly Continuous Functions

In contrast to the original work by Berger, our formalization is limited to unary uniformly continuous functions. It is left as a future work to formalize n-ary uniformly continuous functions in a coinductive setting. While 3-branching trees are used in the unary case, 3^n-branching trees suffice to represent n-ary uniformly continuous functions. We propose to use the type $\mathbf{SD}^n$ which is n-tuple of the signed digit algebra $\mathbf{SD}$ instead of $\mathbf{SD}$ as we used. It can be done by using another framework as Martin-Löf's type theory, or by extending the type system of TCF to dependent types. The type $\mathbf{SD}^n$ is defined as a type dependent on a term n, and the algebras $\mathbf{R}_\alpha$ and $\mathbf{W}$ are also parameterized by n. Terms in T^+ and inductive and coinductive definitions are also to be extended to formalize the idea of Berger.

Appendix A

Proof Assistant Minlog

This appendix describes the proof assistant Minlog. Minlog has been developed by Schwichtenberg and his colleagues for more than 20 years [Sch92, Min]. Minlog is an implementation of TCF, namely, the goal is to make any activity based on TCF doable in Minlog. Proofs are interactively built by a human with some support of automated proof searching. It extracts from constructive proofs computational content based on Kreisel's modified realizability interpretation. Program extraction from non-constructive proofs is also supported by means of the refined A-translation [BBS02, Rat11] and Gödel's dialectica interpretation [Sch08, Tri12]. Terms in T^+ can be exported to general-purpose programming language as Scheme and Haskell. Especially, T^+ is fully supported for the Haskell export which is due to Nordvall [MFS13]. Minlog is implemented in Scheme, and interfaces for human to use Minlog are provided as Scheme procedures.

This appendix consists of two sections. In the first section we discuss technical aspects of the Minlog implementation, being focused on normalization by evaluation, recursion and corecursion operators and inductive and coinductive definitions. In the second section we provide a commentary to work with the Minlog code to the case study given in Section 3.3.

A.1 Implementation

This section describes three selected topics from the implementation of Minlog. The first topic is term normalization of Minlog which is done by means of *normalization by evaluation*, NbE for short, described in Section 2.2.4. The second topic is the internal representation of recursion and corecursion operators. The third topic is the internal representation of inductive and coinductive definitions.

A.1.1 Normalization by Evaluation

Minlog's normalization is done by means of the technique of NbE. The implementation is based on Section 2.2.4. The idea is that a term in T^+ is first sent to its denotation, and then we retrieve from the denotational object a term which is in normal form. The

representation of a semantic object in Minlog is a pair of a type and a native Scheme procedure. We call such a pair an *NbE object*, and call a native Scheme procedure in an NbE object an *NbE value*. Minlog has procedures to deal with NbE objects. Corresponding to term families mentioned in Section 2.2.4 we also introduce procedures to send a term to a term family and vice versa. Proof normalization is also done by NbE through the correspondence between proofs and terms. Note that the extracted term from a proof carries less information than the proof, since non-computational information is thrown away due to the realizability. In order to recover the Curry-Howard correspondence, Minlog regards non-computational parts of formulas in the same way as computational ones in the case of proof normalization. Minlog uses the *NbE algebras* instead of the associated algebras of the defined predicate constants. In contrast to associated algebras, NbE algebras are defined with regard to non-computational universal quantifiers and implications in the same way as computational ones. Moreover, constants as recursion operators are allowed to carry *reproduction data*, which are used to reproduce the formula of the axiom from which these constants are extracted.

In this section, we describe technical details of the implementation of NbE in Minlog system.

A.1.1.1 Term Family

In order to implement NbE, we have to deal with bound variables in in reifying an NbE object of higher type. A term family is used to implement the way to create bound variables. There is a procedure to form a term family.

`(nbe-make-termfam` *type proc*`)`

The internal representation of term families is of the form `('termfam` *type proc*`)` where the type, *type*, is an internal representation of a type and the NbE value, *proc*, is a native Scheme function procedure from a natural number `k` to an internal representation of terms. The following procedure applies a term family $termfam : \mathbb{N} \to \Lambda_\rho$ to a natural number $k : \mathbb{N}$ to get a term of Λ_ρ.

`(nbe-fam-apply` *termfam k*`)`

This procedure applies the NbE value of *termfam* of the form `(lambda (k) ...)` to k. The following procedures sends a term to a term family and vice versa.

`(nbe-term-to-termfam` *term*`)`
`(nbe-extract` *termfam*`)`

The procedure `nbe-term-to-termfam` gives the term family by making a case distinction on the construction of the given term. In any case the type is computed by `term-to-type`. If the outermost construction of *term* is a variable or a constant, the NbE value of the result is `(lambda (k)` *term*`)`. If *term* is an abstraction $\lambda_x t$, the term family is given by

Listing A.1: Abstraction case in `nbe-term-to-termfam`

```
1  (let* ((var (term-in-abst-form-to-var term))
2         (type (var-to-type var))
3         (kernel (term-in-abst-form-to-kernel term)))
4    (nbe-make-termfam
5     (term-to-type term)
6     (lambda (k)
7       (let ((var-k (make-var type k (var-to-t-deg var) (var-to-name var))))
8             (make-term-in-abst-form
9              var-k
10             (nbe-fam-apply
11              (nbe-term-to-termfam
12               (term-subst kernel var (make-term-in-var-form var-k)))
13              (+ 1 k)))))))
```

Listing A.2: Application case in `nbe-term-to-termfam`

```
1  (let ((op (term-in-app-form-to-final-op term))
2        (args (term-in-app-form-to-args term)))
3    (nbe-make-termfam
4     (term-to-type term)
5     (lambda (k)
6      (apply mk-term-in-app-form
7             (map (lambda (x) (nbe-fam-apply (nbe-term-to-termfam x) k))
8                  (cons op args))))))
```

the following code snippet. Let `term` be the given *term*. For the given term $\lambda_x t$, `var` and `kernel` in lines 1 and 3 correspond to x and t, respectively. The variable name x of abstraction is renamed to x_k, i.e., `var-k`, then the body of the function procedure is $\lambda_{x_k}(t[x/x_k]^\infty(k+1))$. If *term* is an application $t\vec{s}$, where $|\vec{s}| \geq 1$, the term family is given by the following code snippet. Let `term` be the given *term*. For the given $t\vec{s}$, `op` and `args` correspond to t and $\vec{s}$, respectively. Then, the body of the function (lines 6–8) constructs $t^\infty(k)s^\infty(k)$.

For a term family r the procedure `nbe-extract` first computes the least natural number k which is larger than any index of a variable in $r(0)$, then the term $r(k)$ is returned.

A.1.1.2 Reflect and Reify

There are two (simultaneous) procedures which translate from a term family into an NbE object and vice versa.

(`nbe-reflect` *termfam*)
(`nbe-reify` *object*)

A term family is sent to an NbE object by `nbe-reflect` and the opposite direction is done by `nbe-reify`. The procedure `nbe-reflect` makes a case distinction on the type of the term family. If it is a type variable or an algebra, the NbE value is the term family, a native Scheme procedure. If it is an arrow type, the NbE value of the result is computed by the

Listing A.3: Arrow type case in `nbe-reflect`

```
1 (lambda (obj)
2   (nbe-reflect (nbe-make-termfam
3                 (arrow-form-to-val-type type)
4                 (lambda (k)
5                   (make-term-in-app-form
6                     (nbe-fam-apply termfam k)
7                     (nbe-fam-apply (nbe-reify obj) k))))))
```

Listing A.4: Algebra case in `nbe-reify`

```
1  (let ((args (nbe-constr-value-to-args value)))
2    (nbe-make-termfam
3     type
4     (lambda (k)
5       (apply mk-term-in-app-form
6              (cons (make-term-in-const-form
7                     (nbe-constr-value-to-constr value))
8              (map (lambda (obj)
9                     (nbe-fam-apply (nbe-reify obj) k))
10                  args))))))
```

following code snippet. Let `type` be the type of the given term family. Let a term family $\lambda_x t^\infty$ be given to `nbe-reflect`. The outermost `(lambda (obj) ...)` is a representation of the abstraction $\lambda_x \ldots$ as a native Scheme procedure. Intuitively the body is given by applying t^∞ to `obj`. Formally, it first constructs a term family $\downarrow$ `obj` by reifying, then the body $t^\infty(k)(\downarrow \texttt{obj})(k)$ is computed. Finally, `nbe-make-termfam` is applied to the body abstracted by `k`. At the end, we apply `nbe-reflect` which is corresponding to $\uparrow$.

The procedure `nbe-reify` takes an NbE object and retrieves a term family. A term family is built by induction on the type of the NbE object. If the type is a type variable, it returns the NbE value of the NbE object. If the type is an algebra, it checks by means of `nbe-constr-value?` whether this NbE object expresses $C\vec{t}$, a constructor C with arguments $\vec{t}$. If this is the case, it makes a term family by the following code snippet. Assume `type` and `value` are the type and the NbE value of the given term family. Assume arguments `args` are prepared as NbE objects corresponding to $\vec{t}$ in line 1. They become reified terms in lines 8–10, then are given to the constructor to be an NbE value. If this is not the case, e.g. it is a variable, then it returns the NbE value of the NbE object. The next case is the arrow type. The following code snippet gives a term family from the NbE object. The procedure `nbe-object-apply` is for function application of NbE objects. Let `type` be the type of the given term family. The variable `var-k` is created with the index `k` which is abstracted by the native Scheme function abstraction. The NbE object obtained by applying the given NbE object `obj` to the NbE object of `var-k` is reified to the term family $\downarrow$ (`obj`($\uparrow$ (`var-k`))). Further applying this term family to `k+1` and abstracting by `var-k`, we obtain the very term in Definition 2.2.47.

Listing A.5: Arrow type case in `nbe-reify`

```
(let ((type1 (arrow-form-to-arg-type type)))
  (nbe-make-termfam
   type
   (lambda (k)
     (let ((var-k (make-var type1 k 1 (default-var-name type1))))
       (make-term-in-abst-form
         var-k (nbe-fam-apply
                 (nbe-reify
                   (nbe-object-apply
                     obj
                     (nbe-reflect (nbe-term-to-termfam
                                    (make-term-in-var-form var-k)))))
                 (+ k 1)))))))
```

A.1.2 Recursion and Corecursion

Recursion and corecursion operators are constants which play an important role in the computational interpretation of inductive and coinductive definitions. We study the implementation of recursion and corecursion operators in Minlog.

It suffices to give (simultaneous) algebras $\vec{\iota}$ and types $\vec{\tau}$ to specify the recursion operator $\mathcal{R}^{\vec{\iota}}_{\vec{\iota},i}$. We have the following procedure to compute a list of the internal representation of recursion operators.

(`arrow-types-to-rec-consts` . *arrow-types*)

For the argument *arrow-types* we give a list of types of the form $\iota_i \to \tau_i$ to specify the recursion operators. The following procedure makes an internal representation of a constant in general.

(`make-const` *obj-or-arity name kind uninst-type tsubst t-deg token-type* . *repro-data*)

The internal representation of constant is just a list consists of `'const`, the identifier for constants, as the head and the arguments given to the above procedure as the rest. The content of *obj-or-arity* in the case of recursion operator is an NbE object of the recursion operator. From the next components, *name*, *kind*, *t-deg* and *token-type* are for the name of the constant, the kind of the computation rule of the constant, the totality degree and the kind of token. In the case of recursion operators, *name* is "`Rec`", *kind* is `'fixed-rules` meaning it is not user changeable, *t-deg* is `1` meaning it is a total functional, and *token-type* is `'const` meaning that this constant is written in a usual function application style, respectively. For *uninst-type* and *tsubst* we give the uninstantiated type of recursion operator and the type substitution which makes the instantiated type. An uninstantiated type is represented by taking all parameter types $\vec{\alpha}$ of algebras $\vec{\iota}$ and all covalue types $\vec{\tau}$ as type variables. The type substitution tells us what substitutes $\vec{\alpha}$ by $\vec{\tau}$. Some type variables may be left. A procedure `arrow-types-to-uninst-recop-types-and-tsubst` computes the

types of $\mathcal{R}^{\vec{\tau}}_{\iota,i}$.

(`arrow-types-to-uninst-recop-types-and-tsubst` . *arrow-types*)

The return value of this procedure is a pair of a list of uninstantiated types of recursion operators and a type substitution. The last argument `repro-data` of `make-const` is optionally given when we normalize a proof. Proof normalization is implemented based on term normalization. Terms in T^+ are not sufficient to represent proofs. Minlog manages missing information by keeping additional information in the internal representation of recursion operator, so that it maintains the Curry-Howard isomorphism.

We explore the procedure `rec-at` which computes the NbE object of recursion operators and see what the NbE object of a recursion operator and how Minlog deals with it.

(`rec-at` *alg-name uninst-recop-type tsubst inst-recop-type*
f rel-constr-names rel-simalg-names-with-uninst-recop-types .
repro-data)

Consider the NbE object of a recursion operator $\mathcal{R}^{\vec{\tau}}_{\iota,i}$ through `rec-at`. The first argument *alg-name*, which is ι_i, determines on which algebra this recursion operator is defined. The type of the recursion operator is represented by a pair of the uninstantiated type of the recursion operator *uninst-recop-type*, and the substitution *tsubst*. For the efficiency reason, the instantiated type *inst-recop-type* is also given. The next argument f carries the number of free variables (or parameters), which is needed in the case of proof normalization, when variables are quantified by the outermost non-computational universal quantifier of an elimination axiom. The following two arguments *rel-constr-names* and *rel-simalg-names-with-uninst-recop-types* depend on the simplification of recursion operators. Considering relevant simultaneous algebra names among all simultaneous algebras, *rel-constr-names* is a list of the names of constructors from the relevant algebras, and *rel-simalg-names-with-uninst-recop-types* is the list of pairs of a relevant algebra name and the uninstantiated type of a recursion operator on the relevant algebra. Assume that we are normalizing a proof involving an elimination axiom which is instantiated as $I\vec{t} \to \ldots \to P\vec{t}$. The last argument plays an important role when this recursion operator is used for proof normalization via the Curry-Howard correspondence. The reproduction data, *repro-data*, are then an implication formula $I\vec{t} \to P\vec{t}$ which is sufficient to recover how the elimination rule is used. The definition of `rec-at` is as follows. Let `inst-recop-type`, `uninst-recop-type` and `repro-data` be the given *inst-recop-type*, *uninst-recop-type* and *repro-data*, respectively. The return value is an NbE object constructed by the instantiated type of the recursion operator and the NbE value. The procedure `nbe-curry` does currying one argument `objs` into arguments of the number depicted by `arity` which is the sum of the number of step cases, parameters and 1. The last 1 in line 4 is for the argument at which the recursion works. We take a closer look at the inside of the first argument of `nbe-curry`, namely, lines 10–11. Let `f` be the given f and suppose that `nbe-for-idps?` is an already computed boolean value. All arguments given to the recursion operator are kept by `objs`. There

Listing A.6: Structure of `rec-at`

```
(let* ((f-plus-s ;number f of free variables plus number s of step types
        (- (length (arrow-form-to-arg-types uninst-recop-type)) 1))
       (s (- f-plus-s f))
       (arity (+ f-plus-s 1))
       (nbe-for-idps? (and (pair? repro-data)
                           (imp-form? (car repro-data)))))
  (nbe-make-object
   inst-recop-type
   (nbe-curry
    (lambda objs
      ... )
    inst-recop-type
    arity)))
```

Listing A.7: NbE value given in `rec-at`

```
(lambda objs
  (let* ((rec-obj (list-ref objs f))
         (rec-val (nbe-object-to-value rec-obj)))
    (cond
     ((or (nbe-fam-value? rec-val) (not REC-UNFOLDING-FLAG))
      ... )
     ((and (nbe-constr-value? rec-val) (not nbe-for-idps?))
      ... )
     ((and (nbe-constr-value? rec-val) nbe-for-idps?)
      ... )
     (else (myerror "rec-at" "value expected" rec-val)))))
```

Listing A.8: Case of term family

```
1  ((or (nbe-fam-value? rec-val) (not REC-UNFOLDING-FLAG))
2   (nbe-reflect
3    (nbe-make-termfam
4     (arrow-form-to-final-val-type inst-recop-type (length objs))
5     (lambda (k)
6       (apply mk-term-in-app-form
7              (make-term-in-const-form ;rec-const
8               (apply alg-name-etc-to-rec-const
9                      alg-name uninst-recop-type tsubst
10                     inst-recop-type f rel-constr-names
11                     rel-simalg-names-with-uninst-recop-types
12                     repro-data))
13             (map (lambda (x) (nbe-fam-apply (nbe-reify x) k))
14                  objs))))))
```

are two bindings: the recursion object `rec-obj` for the argument at which the recurrence works and the recursion value `rec-val` for the NbE value of `rec-obj`. It makes a case distinction by the recursion value. There are four cases. The first case happens if the `rec-val` is the denotation of a term family or `REC-UNFOLDING-FLAG` is false. The flag `REC-UNFOLDING-FLAG` is true by default to enable unfolding of recursion operators, but a user can set it to be false to unblock the unfolding. The second and the third are the case if the `rec-val` is the denotation of a term with a constructor as the outermost operator. A value of such a term is called a *constructor value*. The flag `nbe-for-idps?`, which stands for "NbE for inductively defined predicate constants?", is true if it is for proof normalization of an elimination axiom. It is false if it is for term normalization or for proof normalization of induction axiom by a computational universal quantifier $\forall^{\mathrm{c}}$. Depending on `nbe-for-idps?`, either the second or the third is chosen. If the none of the above cases are applicable, Minlog gives up the execution with putting an error message. From now we study the first two cases to see how term normalization works. The third one deals with parameters to recover the Curry-Howard isomorphism in addition to the second case, in order to perform proof normalization. The following list is of the first case. A value returned by this branch is the NbE object from a term family which is exactly the same as what we are normalizing, namely, this code makes the reproduction of the same thing. This is what we expect for example in the case of normalizing $\mathcal{R}\, x\, \vec{t}$ where x is not formed by any constructor. In the code, the recursion constant is generated by the lines 7–12. The arguments are prepared from `objs` by the lines 13–14 by fixing an index `k` given to the term families.

When the `rec-val` is a constructor value, the recursion operator is converted with respect to the constructor. The following code gives a value for such a case in term normalization. What it computes as the result is the line 9, which is the step term corresponding to the constructor of the value object applied to the arguments. From the name of the constructor `constr-name`, the `step-obj` is chosen. It suffices to compute the arguments called `step-args` which are fed to `step-obj`. From the list of the arguments to the recursion operator, `rel-args` is a sublist of the arguments which are relevant for recurring. The NbE objects for the relevant simultaneous recursion operators are computed for each of

Listing A.9: Case of constructor

```
1  ((and (nbe-constr-value? rec-val) (not nbe-for-idps?))
2   (let*
3       ((constr-name (nbe-constr-value-to-name rec-val))
4        ...
5        (step-obj ... )
6        ...
7        (rel-args ... )
8        ...
9        (step-arg-lists ... )
10       (step-args (apply append step-arg-lists)))
11     (apply nbe-object-app step-obj step-args)))
```

relevant simultaneous algebras. In the code, `recobjs` is their name. The argument types of the constructor determine what the arguments to the `step-obj` are. The arguments are computed and bound to `step-arg-lists` in the code. The internally defined function between lines 3–25 takes two arguments, `rel-arg` and `rel-constr-arg-type` due to `map`, and has three cases. The first case is for parameter arguments of the constructor, namely, the argument type $\rho_{i\nu}$ of the step case is a type variable. The condition between lines 5–7 checks that there is no common algebra both in the list of algebras appear as an argument type of the constructor and the list of the relevant algebras of the recursion operator. The second case is for non-nested recursive arguments of the constructor. The condition between lines 9–14 checks that the value type of the constructor type is an algebra which is among the relevant simultaneous algebras. This case is separated from nested cases because we duplicate the previous input, i.e., `rel-arg`, and the recursive call. The last case is for nested recursive arguments of the constructor. The main point is the use of map operators. Instead of the duplication in the non-nested case, the previous input and the recursive call are paired by the constructor of the product algebra and abstracted to be $\lambda_x\langle x, \mathcal{R}\, x\, \vec{t}\rangle$ which is bound to `ih-fctobjs` in the code. The NbE object of the relevant map operator is on the other hand computed by means of `map-at` and bound to `mapobj`. Finally, it returns the NbE object representing $\mathcal{M}\, N\, (\lambda_x\langle x, \mathcal{R}\, x\, \vec{t}\rangle)_{i<n}$.

Corecursion operators ${}^{\mathrm{co}}\mathcal{R}^{\vec{\tau}}_{\vec{\iota},i}$ are specified by the following procedure.

`(alg-or-arrow-types-to-corec-consts .` *alg-or-arrow-types*`)`

The argument *alg-or-arrow-types* is a list of arrow types $\tau_i \to \iota_i$. When $\vec{\tau}$ is a list of the nulltype, we can give $\vec{\iota}$ instead. The difference from the case of recursion operators is the following.

1. There is `corec-at` to compute the NbE object.

2. The (internal) name of the constant is `"corec"`.

3. There is `alg-or-arrow-types-to-uninst-corecop-types-and-tsubst` to compute the type.

Listing A.10: Three cases in the function definition

```
(step-arg-lists
 (map
  (lambda (rel-arg rel-constr-arg-type)
   (cond
    ((null? ;param-arg-type: take original arg
      (intersection (type-to-alg-names rel-constr-arg-type)
                    rel-simalg-names))
     (list rel-arg))
    ((and (alg-form? (arrow-form-to-final-val-type
                      rel-constr-arg-type))
          (member (alg-form-to-name
                   (arrow-form-to-final-val-type
                    rel-constr-arg-type))
                  rel-simalg-names))
     (list
      rel-arg
      (nbe-object-rec-compose ;first recobj
       (cadr (assoc (alg-form-to-name
                     (arrow-form-to-final-val-type
                      rel-constr-arg-type))
                    rel-simalg-names-to-recobjs-alist))
       step-objs
       rel-arg)))
    (else ;nested recarg-type: use mapop
     ... )))
  rel-args rel-constr-arg-types))
```

Listing A.11: Nested case

```
(else ;nested recarg-type: use mapop
 (let*
     (...
      (ih-fctobjs ... )
      ...
      (mapobj
       (map-at
        rel-constr-arg-type-with-tvars new-tvars
        rel-simalgs rel-simalgs-times-valtypes
        new-val-tvars)))
   (list
    (apply nbe-object-app mapobj rel-arg ih-fctobjs))))
```

Currently, corecursion operators are unchanged during NbE. Instead of NbE, terms involving corecursion operators can be unfolded by the following procedure,

(`undelay-delayed-corec` *term bound*)

where *term* is a term and *bound* is the number of unfoldings. The procedure `corec-at` has only the reproduction case which we saw in `rec-at`. Types of corecursion operators are computed by the procedure `alg-or-arrow-types-to-uninst-corecop-types-and-tsubst` in a similar way as recursion operators.

(`alg-name-etc-to-rec-const`
alg-name uninst-recop-type tsubst inst-recop-type
f rel-constr-names rel-simalg-names-with-uninst-recop-types . repro-data)

A.1.3 Inductive and Coinductive Definitions

Inductive definitions are taken care of by the procedure `add-ids` in Minlog.

(`add-ids` *idpc-names-with-arities-and-opt-alg-names .*
clause-strings-with-opt-names)

In order to define (simultaneous) inductive predicates $\vec{I}$, we have to specify the names of the predicates, the arities and the clause formulas. For *idpc-names-with-arities-and-opt-alg-names* we give a list of the name of the predicate, the arity and optionally the associated type by an algebra name. There is a naming convention `algI` for a predicate name `I` used to define a new algebra automatically by the name `algI`. One can also specify an existing algebra name instead. For clause formulas we give a list of an introduction axiom and optionally its name as *clause-strings-with-opt-names*. Most of the businesses of `add-ids` is to validate the given definition. For instance, if an existing algebra name is provided for the associated algebra, it checks if the computed associated algebra from clause formulas matches indeed. A coinductively defined predicate can be generated by dualizing an inductively defined predicate by `add-co`.

(`add-co` *idpc-name . opt-prim-prod-flag*)

This function forms the closure axiom of a coinductive predicate from the introduction axioms of a given inductive predicate *idpc-name*. If *opt-prim-prod-flag* is not given or true, the axiom is formed with the primitive existential quantifier and conjunction, which sometimes are preferable for efficiency reasons. If the flag is false, i.e., `#f`, inductively defined existential quantifier and conjunction are used.

We give an example of proof normalization to sketch how the third case of `rec-at`, at line 9 and 10 in Listing A.7, works for inductively defined predicates. A recursion operator can encode an elimination axiom due to reproduction data. We use special algebras so-called *NbE algebras* as well as reproduction data to let recursion operators and also constructors to carry free variables needed to restore the proof.

Example A.1.1 (Proof normalization involving the inductive predicate Even)**.** Recall the predicate Even defined in Example 2.3.11 with a proper decoration and consider the following proof of $\mathrm{Even}(n+4) \to^{\mathrm{c}} P0 \to^{\mathrm{c}} \forall_n^{\mathrm{nc}}(\mathrm{Even}\, n \to^{\mathrm{c}} P\, n \to^{\mathrm{c}} P(n+2)) \to^{\mathrm{c}} P\, 4$. We refer to $P\,0$ and $\forall_n^{\mathrm{nc}}(\mathrm{Even}\, n \to^{\mathrm{c}} P\, n \to^{\mathrm{c}} P(n+2))$ by A_0 and A_1, respectively, and abuse the name of axioms for their conclusion in the proofs.

$$\dfrac{\dfrac{\dfrac{\mathrm{Even}^-\quad n+4}{\mathrm{Even}(n+4) \to^{\mathrm{c}} A_0 \to^{\mathrm{c}} A_1 \to^{\mathrm{c}} P(n+4)} \quad \dfrac{\mathrm{Even}_1^+\quad n+2 \qquad \dfrac{\mathrm{Even}_1^+\quad n\quad [\mathrm{Even}\, n]^u}{\mathrm{Even}(n+2)}}{\mathrm{Even}(n+4)}}{A_0 \to^{\mathrm{c}} A_1 \to^{\mathrm{c}} P(n+4)}}{\mathrm{Even}\, n \to^{\mathrm{c}} A_0 \to^{\mathrm{c}} A_1 \to^{\mathrm{c}} P(n+4)}\ (\to^{\mathrm{c}})^+, u$$

The NbE algebra of Even is $\iota := \mu_\xi^{\mathrm{NbE}}(C_0^\xi, C_1^{\mathbf{N}\to\xi\to\xi})$ where the $\mathbf{N}$ in the second constructor type comes from the $\forall_n^{\mathrm{nc}}$ in Even_1^+ and is treated as a parameter. From this special algebra Minlog generates a recursion operator $\mathcal{R}_\iota^\tau : \mathbf{N} \to \iota \to \tau \to (\mathbf{N} \to \iota \to \tau \to \tau) \to \tau$. The conversion rules are $\mathcal{R}_\iota^\tau\, n\, C_0\, M\, N \mapsto M$ and $\mathcal{R}_\iota^\tau n(C_1\, m\, u)M\, N \mapsto N\, m\, u(\mathcal{R}_\iota^\tau\, m\, u\, M\, N)$ where the first argument n is considered as a parameter. The number of the parameters is given as the argument f of `rec-at`. The term representation of the proof is $\lambda_u(\mathcal{R}_\iota(n+4)(C_1(n+2)(C_1\, n\, u)))$. Each constructor C_i carries the number i and the inductively defined predicate constant Even as the reproduction data. The recursion operator $\mathcal{R}_\iota$ carries the formula $\mathrm{Even}(n+4) \to P(n+4)$ instead. The normal form of the term is $\lambda_{u,w_0,w_1}(w_1(n+2)(C_1\, n\, u)(w_1\, n\, u(\mathcal{R}_\iota\, n\, u\, w_0\, w_1)))$ which represents the following normalized proof.

$$\dfrac{\dfrac{\dfrac{[A_1]^{w_1}\quad n+2 \qquad \dfrac{\mathrm{Even}_1^+\quad n\quad [\mathrm{Even}\, n]^u}{\mathrm{Even}(n+2)} \quad \dfrac{[A_1]^{w_1}\quad n\quad [\mathrm{Even}\, n]^u \qquad \dfrac{\mathrm{Even}^-\quad n\quad [\mathrm{Even}\, n]^u\quad [A_0]^{w_0}\quad [A_1]^{w_1}}{P\, n}}{P(n+2)}}{P(n+4)}}{A_1 \to^{\mathrm{c}} P(n+4)}\ (\to^{\mathrm{c}})^+, w_1}{A_0 \to^{\mathrm{c}} A_1 \to^{\mathrm{c}} P(n+4)}\ (\to^{\mathrm{c}})^+, w_0}{\mathrm{Even}\, n \to^{\mathrm{c}} A_0 \to^{\mathrm{c}} A_1 \to^{\mathrm{c}} P(n+4)}\ (\to^{\mathrm{c}})^+, u$$

If there are free variables in a competitor predicate of an elimination axiom or a predicate parameter of an inductive predicate, the types of axioms are given differently to carry all free variables as parameters.

A.2 Commentary to `cauchysds.scm`

All of the case studies in Chapter 3 are formalized and running in the Minlog system [Min]. To complement the proofs illustrated rather informally, we take a look at Minlog proof scripts on Cauchy reals and SDSs in Section 3.3 with focus on Proposition 3.3.7. In the Minlog distribution, this is in `examples/analysis/cauchysds.scm`.

Using standard procedures to define algebras, axioms, predicates and etc., the corresponding Scheme global variables are modified to store internal representations of definitions.

Listing A.12: Loading libraries

```
1  ;(load "~/minlog/init.scm")
2
3  ;; Contents
4  ;; 1. A Cauchy real to an SDS. (PropA)
5  ;; 2. An SDS to a Cauchy real. (PropB)
6  ;; 3. Experiments.
7
8  (set! COMMENT-FLAG #f)
9  (libload "nat.scm")
10 (libload "numbers.scm")
11 (exload "analysis/real.scm")
12 (set! COMMENT-FLAG #t)
```

Proofs in Minlog are interactively built by proof tactics commands, which in fact manipulate the goal and context stored in a global variable. Computational content from proofs is mechanically extracted. The program in T^+ can be exported to general-purpose programming languages as Scheme and Haskell. The imlementation of the feature for Haskell is due to Nordvall [MFS13]. In this section the file path refers to a relative path from the Minlog directory `minlog/` of the distribution.

A.2.1 Definitions

The beginning of the code is to load the necessary libraries. The first line can be uncommented to start up the Minlog system. In line 9, the library for natural number is loaded. The entity of a library is a Scheme file placed at `lib/`. In line 10, the library for numbers including rational, (type-1) real and complex numbers is loaded. We use positive numbers, integers and rational numbers out of it. In line 11, an existing case study on real numbers is loaded. Although we do not use existing real numbers from the library, we make use of a proof tactic, called `ord-field-simp-bwd`, defined in this file for rational number reasoning. In lines 8 and 12, the flag to allow/disallow Minlog to output a message is switched, so that Minlog is silent except error messages during loading libraries.

It starts from declaring variable names. In lines 15 and 16, an existing variable name `M` is removed. It is used in `lib/numbers.scm` for a variable name ranging over $\mathbf{Z} \to \mathbf{N}$ which we are going to use for another purpose. In line 18, a type variable `r` is declared for the type ρ of abstract real numbers. In line 19 and 20, it declares that the variable `x` ranges over `r` after removing the existing declaration. We define the algebra **SD** of the signed digits and the programmable constant `SDToInt` of type $\mathbf{SD} \to \mathbf{Z}$ with the computation rules. By means of `add-alg`, **SD** is defined to be $\mu_\xi(\mathsf{L}^\xi, \mathsf{M}^\xi, \mathsf{R}^\xi)$. The totality predicate of an algebra is derived by `add-totality`. A programmable constant is declared by using `add-program-constant`. It is required to give a name and a type, where a type can also be constructed by the arrow type "`=>`". Then, the computation rules are defined by `add-computation-rule`. It is obvious that the programmable constant `SDToInt` has a computational content to coerce a signed digit to an integer. Here, the types of "`0`" and "`1`" in the second arguments are inferred by Minlog. In line 26, `1` is a positive number which is given to the constructor

Listing A.13: Variable names declaration

```
14 ;; 1. A Cauchy real to an SDS
15 (remove-var-name "M") ;will be used as constructor for sd
16 (remove-token "M")
17
18 (add-tvar-name "r")
19 (remove-var-name "x")
20 (add-var-name "x" (py "r"))
```

Listing A.14: Algebra **SD**

```
22 (add-alg "sd" '("L" "sd") '("M" "sd") '("R" "sd"))
23 (add-totality "sd")
24
25 (add-program-constant "SDToInt" (py "sd=>int"))
26 (add-computation-rule (pt "SDToInt L") (pt "IntN 1"))
27 (add-computation-rule (pt "SDToInt M") (pt "0"))
28 (add-computation-rule (pt "SDToInt R") (pt "1"))
```

`IntN` of type `pos=>int`. On the other hand in lines 27 and 28, Minlog recognizes that `0` and `1` are of type `int`, but neither of `nat` nor `pos`. Due to the nature of TCF, everything is partial in principle. Therefore, one has to prove the totality if it is necessary. In line 30, there are three commands, `set-goal`, `term-to-totality-formula` and `pt`. The last one stands for "parse term." It first parses the string to generate an internal term representation. The totality statement of `SDToInt`, namely, $\forall_d^{\mathrm{nc}}(T_{\mathbf{SD}}d \to T_{\mathbf{Z}}(\mathtt{SDToInt}\, d))$, is given by `term-to-totality-formula`. Finally, this formula is set to be the goal to prove. In line 31, a universally quantified variable and a premise are assumed by the command `assume` with the specified names. In line 32, $(T_{\mathbf{SD}})^-$ is used to the current goal. Since there are three step cases, the Minlog generates three new goals, which are proved in lines 33–35, 36–37 and 38–40, respectively. After using `ng`, standing for "normalizing goal," the first goal becomes $T_{\mathbf{Z}}(\mathtt{N}\,1)$. The argument `#t` is used in order to normalize only the goal, but no context. Applying $(T_{\mathbf{Z}})_0^+$, then prove $T_{\mathbf{P}}\,1$ by using $(T_{\mathbf{P}})_0^+$. The second goal is $T_{\mathbf{Z}}\,0$ which is proven by $(T_{\mathbf{Z}})_1^+$. The last one is by $(T_{\mathbf{Z}})_2^+$, then prove $T_{\mathbf{P}}\,1$ as in the fist case. When all sub goals have been closed, the Minlog tells us that the proof is finished. The internally built proof can be saved by the `save` command with giving a name. One can access the proof object later by the name. In line 44, we define the algebra **I** which works in a similar way as $\mathbf{L_{SD}}$. Although this algebra can be automatically extracted from the definition of an inductively defined predicate in line 51, we manually define it here in order to give user specified names of the algebra and its constructors. In lines 46–49, the constants in Definition 3.3.2, 3.3.3 and 3.3.4 are given. If a programmable constant involves type variables, type arguments have to be given as in the line 52. In lines 51–55, the coinductive predicate ${}^{\mathrm{co}}I$ in Definition 3.3.6 is given. It first defines the inductive predicate I of arity (ρ) with specified associated algebra **I**. In lines 52–53, the clause formulas are given to define I to be $\mu_X(X\,\mathrm{Z}, \forall_x^{\mathrm{nc}}\forall_d^{\mathrm{c}}(Xx \to X(\frac{x+d}{2})))$ with the names `InitI` and `GenI` for I_0^+ and I_1^+, respectively. A string can represent a formula in the following syntax: `allnc` for $\forall^{\mathrm{nc}}$,

Listing A.15: Totality of `SDToInt`

```
30 (set-goal (term-to-totality-formula (pt "SDToInt")))
31 (assume "sd^" "Td")
32 (elim "Td")
33 (ng #t)
34 (use "TotalIntIntNeg")
35 (use "TotalPosOne")
36 (ng #t)
37 (use "TotalIntIntZero")
38 (ng #t)
39 (use "TotalIntIntPos")
40 (use "TotalPosOne")
41 ; Proof finished.
42 (save "SDToIntTotal")
```

Listing A.16: Inductively defined predicate I

```
44 (add-alg "iv" '("II" "iv") '("C" "sd=>iv=>iv"))
45
46 (add-program-constant "Elem" (py "r=>rat=>nat=>boole"))
47 (add-program-constant "Z" (py "r")) ; zero
48 (add-program-constant "Av" (py "r=>sd=>r")) ;average
49 (add-program-constant "Va" (py "r=>sd=>r")) ;inverse of average
50
51 (add-ids (list (list "I" (make-arity (py "r")) "iv"))
52    '("I(Z r)" "InitI")
53    '("allnc x^ all sd(I x^ -> I((Av r)x^ sd))" "GenI"))
54
55 (add-co "I")
```

`all` for $\forall^{\mathrm{c}}$, `->` for $\to^{\mathrm{c}}$ and `-->` for $\to^{\mathrm{nc}}$. A variable followed by a hat "`^`" means a partial variable, whereas a variable without it means a total variable. Apart from declared variable names, the algebra name can be used as a variable name, e.g. `sd` in the line 53. In line 55, the coinductively defined predicate ${}^{\mathrm{co}}I$ is derived from I. The last part of the definitions is the axioms of the abstract theory of real numbers. By means of `add-global-assumption`, we postulates a formula to be an axiom with a specified name. In addition to the syntax which we have seen, there are `<=` for "less than or equal" on rational numbers, `#` standing for the constructor of $\mathbf{Q}$, and `PosToNat` is for the type coercion. Also some arithmetic on rational numbers, `+`, `-`, `*` and `/`, are there. For example, `IntN 1#4` stands for $\frac{-1}{4}$.

A.2.2 Proofs

We prove Lemma 3.2.7 and 3.2.8. In line 60, `ori` stands for a logical connective "or," which is parsed to be one of $\vee^{\mathrm{d}}$, $\vee^{\mathrm{r}}$, $\vee^{\mathrm{l}}$, $\vee^{\mathrm{u}}$ depending on the computational content from the left and the right hand formulas. In this case, it becomes $\vee^{\mathrm{u}}$ since the inequality is non-computational from its definition, thus the statement is $\forall^{\mathrm{c}}_a(a \le 0 \vee^{\mathrm{u}} 0 \le a)$.

The first `cases` does case distinction on the construction of the rational number `rat` to yield `i` and `p` such that `rat` = `i#p`. In fact this proof tactic is a restricted form of the elimination axiom. The second `cases` does the same for the integer `i`. There are three

Listing A.17: Axioms

```
132 ;;Axioms for PropA
133 (add-global-assumption
134  "AxRealLeft"
135  "all x^,a(a<=(IntN 1#4) -> (Elem r)x^ a(PosToNat 2) ->
136            (Elem r)x^(IntN 1#2)(PosToNat 1))")
137
138 (add-global-assumption
139  "AxRealMiddle"
140  "all x^,a((IntN 1#4)<=a -> a<=(1#4) -> (Elem r)x^ a(PosToNat 2) ->
141            (Elem r)x^(0#2)(PosToNat 1))")
142
143 (add-global-assumption
144  "AxRealRight"
145  "all x^,a((1#4)<=a -> (Elem r)x^ a(PosToNat 2) ->
146            (Elem r)x^(1#2)(PosToNat 1))")
147
148 (add-global-assumption
149  "AxAvVaIdent"
150  "all x^,sd((Elem r)x^(SDToInt sd#2)(PosToNat 1) ->
151              x^ eqd(Av r)((Va r)x^ sd)sd)")
152
153 (add-global-assumption
154  "AxVaIntro"
155  "all x^,sd,a,n((Elem r)x^(SDToInt sd#2)(PosToNat 1) ->
156    (Elem r)x^ a(Succ n) -> (Elem r)((Va r)x^ sd)(2*a-SDToInt sd)n)")
```

cases since `i` is either positive, zero or negative. In any case, it is easy to determine whether `rat <= 0` or `0 <= rat`. The `assume` tactic is for an introduction rule of implication and the universal quantifier. New assumptions come with specified names. In lines 64, 67 and 70, the proof tactic `intro` applies the introduction axiom to the goal. Minlog finds an appropriate predicate $\vee^{\mathrm{u}}$ from the goal formula, and chooses $(\vee^{\mathrm{u}})^{+}_{0}$ or $(\vee^{\mathrm{u}})^{+}_{1}$ by the given number. The `use` tactic applies the specified assumption, axiom, or theorem to the goal. The name "`Truth-Axiom`" refers to the truth $\mathbf{T}$, which is available as an axiom. One can save the constructed proof by `save` command, and also can use it later as a theorem. The following is the proof of Lemma 3.2.8. In the lines 78, 81 and so on, `assert` does the implication elimination. Assume that the formula `A` is given as an argument and the goal is `B`, then two new goals, `B` and `B -> A`, are then required to be proven. In the line 79, the proof tactic `ord-field-simp-bwd` solves the equality of rational numbers, `a=a-b+b`. This is one thing which we need from the external file `examples/analysis/real.scm`. The proven lemma can be referred to by the name as it is done in the line 82. The `elim` tactic uses the elimination axiom of an inductively defined predicate. In the line 84, `elim` is called with the assumption formula `a-b<=0 oru 0<=a-b`, then internally it applies $(\vee^{\mathrm{u}})^{-}$ to the goal and yields step formulas which become new sub goals. The `drop` command is to make the specified assumption invisible in the context. It just changes the display, but there is no mathematical effect. The name `RatPlusLe2` is the following global assumption which is postulated in `examples/analysis/real.scm`.

$$\forall_{a_1,a_2,b_1,b_2}(a_1 \le a_2 \to b_1 \le b_2 \to a_1 + a_2 \le b_1 + b_2).$$

Listing A.18: Lemma NegOrPos

```
59 ; "NegOrPos"
60 (set-goal "all rat(rat<=0 ori 0<=rat)")
61 (cases)
62 (cases)
63 (assume "pos0" "pos1")
64 (intro 1)
65 (use "Truth-Axiom")
66 (assume "pos0")
67 (intro 0)
68 (use "Truth-Axiom")
69 (assume "pos0" "pos1")
70 (intro 0)
71 (use "Truth-Axiom")
72 ; Proof finished.
73 (save "NegOrPos")
```

The `simp` tactic rewrites the goal by the given equality. Now the following proof is clear. In the proof of Lemma 3.2.9 named `"StandardSplit"` in the proof script, we use `inst-with-to` which is a tactic for the forward reasoning. The first argument is an assumption and the next arguments are used to eliminate $\forall$ and $\to$, then the obtained new assumption is named by the last argument. For example the line 111 makes from SplitAtRational, a, and $-\frac{1}{4}$ a new assumption formula $a \le -\frac{1}{4} \vee^{\mathrm{u}} -\frac{1}{4} \le a$ with the name `"at -1#4"`. By means of similar tactics we can prove the lemma `"SplitProp"` in line 173. The following code snippet is the proof script of Proposition 3.3.7. The goal formula is set to be $\forall_x^{\mathrm{nc}}(\forall_n^{\mathrm{c}}\exists_a^{\mathrm{l}} x \in \mathbb{I}_{a,n} \to {}^{\mathrm{co}}Ix)$ in line 209. We assume x and $\forall_n^{\mathrm{c}}\exists_a^{\mathrm{l}} x \in \mathbb{I}_{a,n}$ as Qx in line 210, then use ${}^{\mathrm{co}}I^+$ for the competitor Qx to prove ${}^{\mathrm{co}}Ix$. It suffices to prove the costep formula $\forall_x^{\mathrm{nc}}(Qx \to x \text{ eqd } \mathrm{Z} \vee^{\mathrm{r}} \exists_{x_0}^{\mathrm{r}}\exists_d^{\mathrm{d}}(({}^{\mathrm{co}}Ix_0 \vee^{\mathrm{d}} Qx_0) \wedge^{\mathrm{l}} x \text{ eqd } \frac{x_0+d}{2}))$. Assume x_1 and Qx_1 in line 290, then we use $(\vee^{\mathrm{r}})_1^+$ to prove the right disjunct in line 291. Applying the split property lemma to x_1 and Qx_1, we have $\exists_d^{\mathrm{l}}(x_1 \in \mathbb{I}_d)$ in the context by the name `"SP"`. This is done by the command `inst-with-to`. We assert the same formula by `assert`. Invoking `(assert` A`)`, the current goal B turn into the two subgoals A and $A \to^{\mathrm{c}} B$. Although It looks logically non-sense, the application of the identity lemma in line 223 fine-tunes the computational content. See Example 2.5.28. The command `elim` in line 224 is an application of $(\exists^{\mathrm{l}})^-$ for the formula `"SP"`. The goal is then $\forall_d^{\mathrm{c}}(x_1 \in \mathbb{I}_d \to^{\mathrm{nc}} \exists_x^{\mathrm{r}}(\exists_{d_0}^{\mathrm{d}}(({}^{\mathrm{co}}Ix \vee^{\mathrm{d}} \forall_n^{\mathrm{c}}\exists_a^{\mathrm{l}} x \in \mathbb{I}_{a,n}) \wedge^{\mathrm{l}} x_1 \text{ eqd } \frac{x+d_0}{2}))$. We assume d and $x_1 \in \mathbb{I}_d$ as `HElem` in line 225, and use $(\exists^{\mathrm{r}})^+$ with $2x_1 - d$ and also $(\exists^{\mathrm{d}})^+$ with d. The command `split` turns the goal of the form $A \wedge B$ into the two subgoals A and B. The lines 230–238 is to prove ${}^{\mathrm{co}}Ix \vee^{\mathrm{d}} \forall_n^{\mathrm{c}}\exists_a^{\mathrm{l}} x \in \mathbb{I}_{a,n}$. We go to the right disjunct in line 230 and assume n in the next line. The formula Qx_1 in the context is instantiated to $\exists_a^{\mathrm{l}} x_1 \in \mathbb{I}_{a,n+1}$ as `Hex` in line 232, then we use $(\exists^{\mathrm{l}})^-$ for this formula in line 233. Assume a as `a` and $x_1 \in \mathbb{I}_{a,n+1}$ as `HElem2`. Using $(\exists^{\mathrm{l}})^+$ with $2a - d$, we get the new goal $2x_1 - d \in \mathbb{I}_{2a-d,n}$. By the axiom `AxVaIntro` it suffices to show $x_1 \in \mathbb{I}_d$ and $x_1 \in \mathbb{I}_{a,n+1}$ which are `HElem` and `HElem2`, respectively. The right conjunct is x_1 eqd $\frac{2x_1-d+d}{2}$ which is proven by `AxAvVaIntro` and `HElem`. The proof is done and saved as `PropA`.

Listing A.19: Lemma SplitAtRational

```
; "SplitAtRational"
(set-goal "all a,b(a<=b ori b<=a)")
(assume "a" "b")
(assert "a=a-b+b")
 (ord-field-simp-bwd)
(assume "Eq")
(assert (pf "a-b<=0 ori 0<=a-b"))
 (use "NegOrPos")
(assume "NegOrPos inst")
(elim "NegOrPos inst")
(drop "NegOrPos inst")
(assume "Hneg")
(intro 0)
(assert (pf "(a-b)+b<=0+b"))
 (use "RatPlusLe2")
 (use "Hneg")
 (use "Truth-Axiom")
(assume "H0")
(simp "Eq")
(use "H0")
(assume "Hpos")
(intro 1)
(assert (pf "0+b<=(a-b)+b"))
 (use "RatPlusLe2")
 (use "Hpos")
 (use "Truth-Axiom")
(assume "H0")
(simp "Eq")
(use "H0")
; Proof finished.
(save "SplitAtRational")
```

Listing A.20: Lemma Standard Split

```
; "Standard Split"
(set-goal "all a(a <= (IntN 1#4) ori
      ((IntN 1#4) <= a & a <= (IntP 1#4)) ori (IntP 1#4) <= a)")
(assume "a")
(inst-with-to "SplitAtRational" (pt "a") (pt "IntN 1#4") "at -1#4")
(elim "at -1#4")
(assume "a<=-1#4")
(intro 0)
(use "a<=-1#4")
(assume "-1#4<=a")
(inst-with-to "SplitAtRational" (pt "a") (pt "IntP 1#4") "at 1#4")
(elim "at 1#4")
(assume "a<=1#4")
(intro 1)
(intro 0)
(split)
(use "-1#4<=a")
(use "a<=1#4")
(assume "1#4<=a")
(intro 1)
(intro 1)
(use "1#4<=a")
; proof finished
(save "StandardSplit")
```

Listing A.21: Split Property Lemma

```
172 ; Split Property Lemma for Abstract Reals
173 (set-goal "allnc x^(all n exi a((Elem r)x^ a n) ->
174                     exi sd (Elem r)x^(SDToInt sd#2)1)")
```

Listing A.22: PropA

```
208 ; "PropA"
209 (set-goal (pf "allnc x^(all n exi a((Elem r)x^ a n) -> CoI x^)"))
210 (assume "x^" "Q x")
211 (coind "Q x")
212 ;; ?_3:allnc x^(
213 ;;       all n exl a (Elem r)x^ a n ->
214 ;;       x^ eqd(Z r) orr
215 ;;       exr x^0
216 ;;        ex sd(
217 ;;         (CoI x^0 ord all n exl a (Elem r)x^0 a n) andl x^ eqd(Av r)x^0 sd))
218 (assume "x^1" "Q x1")
219 (intro 1)
220 (inst-with-to "SplitProp" (pt "x^1") "Q x1" "SP")
221 (assert (pf "exl sd (Elem r)x^1(SDToInt sd#2)(PosToNat 1)"))
222  (use "SP")
223 (use "Id")
224 (elim)
225 (assume "sd" "HElem")
226 (intro 0 (pt "(Va r)x^1 sd"))
227 (ex-intro (pt "sd"))
228 (split)
229 ;CoI
230 (intro 1)
231 (assume "n")
232 (inst-with-to "Q x1" (pt "Succ n") "Hex")
233 (elim "Hex")
234 (assume "a" "HElem2")
235 (intro 0 (pt "2*a-(SDToInt sd)"))
236 (use "AxVaIntro")
237 (use "HElem")
238 (use "HElem2")
239 ; x^1 eqd ...
240 (use "AxAvVaIdent")
241 (use "HElem")
242 ; proven
243 (save "PropA")
```

Listing A.23: Program extraction

```
282 ;; 3. Experiments
283 ;; 3.1. Cauchy Sequence -> SDS
284 (define eterm-a
285   (proof-to-extracted-term (theorem-name-to-proof "PropA")))
286 (define neterm-a (nt eterm-a))
```

Listing A.24: Animating

```
288 (animate "SplitProp")
289 (animate "NegOrPos")
290 (animate "SplitAtRational")
291 (animate "StandardSplit")
```

A.2.3 Program Extraction

We extract a program from the proof of `PropA` and experiment it. The command `theorem-name-to-proof` gives the proof object of `PropA`. Program extraction is done by `proof-to-extracted-term`. The extracted program is named as `eterm-a`. In the next line, let `neterm-a` be the normal form of `eterm-a`. The computational contents from the lemmas used in the proof of `PropA` should be activated. The `save` command also define the constant corresponding to the computational content of the proof to save. The `animate` command activates the computational content of the constant through program extraction. The following line 294 normalizes the term `neterm-a` using also the animated computational contents and prints out the result with the case expression by `ppc` which stands for pretty printing with cases. Now we activate the computational content from the identity lemma as well. The following program constant `sqrt` is a hand-implemented function computing the square root of the given rational number. The result is of type $\mathbf{N} \to \mathbf{Q}$. For instance, the following code computes the SDS representation of $\sqrt{\frac{1}{2}}$. Programs in T^+ involving corecursion operators can be exported to Haskell. The file `examples/analysis/simpreal.scm` is required for numeric terms we used so far. Then, the next line outputs the Haskell translation of the extracted programs in `cauchysds.scm` and the hand implemented square root program to the file `cauchysds.hs` in the working directory.

Listing A.25: Extracted program

```
;; extracting a program from PropA
(ppc (nt neterm-a))
;; [as0]
;;  (CoRec (nat=>rat)=>iv)as0
;;  ([as1]
;;    Inr[let sd2
;;         [let a2
;;          (as1(Succ(Succ Zero)))
;;          [case (a2-(IntN 1#4))
;;           (k3#p4 ->
;;           [case k3
;;             (p5 ->
;;             [case (a2-(1#4))
;;               (k6#p7 -> [case k6 (p8 -> R) (0 -> M) (IntN p8 -> M)])])
;;             (0 -> L)
;;             (IntN p5 -> L)])]]
;;         (sd2@(InR nat=>rat iv)([n3]2*as1(Succ n3)-SDToInt sd2))])
```

Listing A.26: Animating the identity lemma

```
(animate "Id")
```

Listing A.27: Square root

```
;; a rational number to the square root of it
(add-program-constant "sqrt" (py "rat=>nat=>rat"))
(add-program-constant "sqrtaux" (py "rat=>nat=>rat"))
(add-computation-rule "sqrtaux a Zero" "Succ Zero")
(add-computation-rule "sqrtaux a (Succ n)"
          "((sqrtaux a n) + (a / (sqrtaux a n)))/2")
(add-computation-rule "sqrt a n" "sqrtaux a (Succ n)")
```

Listing A.28: Unfolding corecursion operators

```
(pp (nt (undelay-delayed-corec
         (make-term-in-app-form neterm-a (pt "sqrt (1#2)"))
         5)))
;; C R(C R(C M(C L(C R((CoRec (nat=>rat)=>iv) ... )))))
```

Listing A.29: Haskell translation

```
;; 3.3. Haskell translation
(load "~/minlog/examples/analysis/simpreal.scm")
(terms-to-haskell-program "cauchysds.hs"
        (list (list neterm-a "cauchysds")
        (list neterm-b "sdscauchy")
        (list (pt "sqrt") "rattosqrt")))
```

Bibliography

[AA99] Andreas Abel and Thorsten Altenkirch. A predicative strong normalisation proof for a lambda-calculus with interleaving inductive types. In Thierry Coquand, Peter Dybjer, Bengt Nordström, and Jan M. Smith, editors, *Types for Proofs and Programs, International Workshop TYPES'99, Lökeberg, Sweden, June 12-16, 1999, Selected Papers*, volume 1956 of *Lecture Notes in Computer Science*, pages 21–40. Springer, 1999.

[Abe07] Andreas Abel. Mixed inductive/coinductive types and strong normalization. In Zhong Shao, editor, *APLAS*, volume 4807 of *Lecture Notes in Computer Science*, pages 286–301. Springer, 2007.

[BB85] Errett Bishop and Douglas Bridges. *Constructive Analysis*, volume 279 of *Grundlehren der mathematischen Wissenschaften*. Springer Verlag, Berlin, Heidelberg, New York, 1985.

[BBLS06] Ulrich Berger, Stefan Berghofer, Pierre Letouzey, and Helmut Schwichtenberg. Program extraction from normalization proofs. *Studia Logica*, 82:27–51, 2006.

[BBS+98] Holger Benl, Ulrich Berger, Helmut Schwichtenberg, Monika Seisenberger, and Wolfgang Zuber. Proof theory at work: Program development in the Minlog system. In W. Bibel and P.H. Schmitt, editors, *Automated Deduction – A Basis for Applications*, volume II: Systems and Implementation Techniques of *Applied Logic Series*, pages 41–71. Kluwer Academic Publishers, Dordrecht, 1998.

[BBS02] Ulrich Berger, Wilfried Buchholz, and Helmut Schwichtenberg. Refined program extraction from classical proofs. *Annals of Pure and Applied Logic*, 114:3–25, 2002.

[BC85] Joseph L. Bates and Robert L. Constable. Proofs as programs. *ACM Transactions on Programming Languages and Systems*, 7(1):113–136, January 1985.

[Ber93] Ulrich Berger. Program extraction from normalization proofs. In M. Bezem and J.F. Groote, editors, *Typed Lambda Calculi and Applications*, volume 664 of *LNCS*, pages 91–106. Springer Verlag, Berlin, Heidelberg, New York, 1993.

[Ber02] Stefan Berghofer. Program extraction in simply-typed higher order logic. In Geuvers and Wiedijk [GW03], pages 21–38.

[Ber05] Ulrich Berger. Uniform heyting arithmetic. *Ann. Pure Appl. Logic*, 133(1-3):125–148, 2005.

[Ber09] Ulrich Berger. From coinductive proofs to exact real arithmetic. In E. Grädel and R. Kahle, editors, *Computer Science Logic*, LNCS, pages 132–146. Springer Verlag, Berlin, Heidelberg, New York, 2009.

[Ber10] Ulrich Berger. Realisability for induction and coinduction with applications to constructive analysis. *J. UCS*, 16(18):2535–2555, 2010.

[Ber11] Ulrich Berger. From coinductive proofs to exact real arithmetic: theory and applications. *Logical Methods in Comput. Sci.*, 7(1):1–24, 2011. `http://www.lmcs-online.org/ojs/viewarticle.php?id=704`.

[BES03] Ulrich Berger, Matthias Eberl, and Helmut Schwichtenberg. Term rewriting for normalization by evaluation. *Information and Computation*, 183:19–42, 2003.

[BH08] Ulrich Berger and Tie Hou. Coinduction for exact real number computation. *Theory Comput. Syst.*, 43(3-4):394–409, 2008.

[Bis67] Errett Bishop. *Foundations of Constructive Analysis*. McGraw-Hill, New York, 1967.

[BMSS11] Ulrich Berger, Kenji Miyamoto, Helmut Schwichtenberg, and Monika Seisenberger. Minlog — a tool for program extraction supporting algebras and coalgebras. In Andrea Corradini, Bartek Klin, and Corina Cîrstea, editors, *CALCO*, volume 6859 of *Lecture Notes in Computer Science*, pages 393–399. Springer, 2011.

[BN02] Stefan Berghofer and Tobias Nipkow. Executing higher order logic. In P. Callaghan, Z. Luo, J. McKinna, and R. Pollack, editors, *Types for Proofs and Programs (TYPES 2000)*, volume 2277 of *LNCS*, pages 24–40. Springer Verlag, Berlin, Heidelberg, New York, 2002.

[BS91] Ulrich Berger and Helmut Schwichtenberg. An inverse of the evaluation functional for typed λ-calculus. In R. Vemuri, editor, *Proceedings 6'th Symposium on Logic in Computer Science (LICS '91)*, pages 203–211. IEEE Computer Society Press, Los Alamitos, 1991.

[BS10] Ulrich Berger and Monika Seisenberger. Proofs, programs, processes. In F. Ferreira et al., editors, *Proceedings CiE 2010*, volume 6158 of *LNCS*, pages 39–48. Springer Verlag, Berlin, Heidelberg, New York, 2010.

[BS12] Ulrich Berger and Monika Seisenberger. Proofs, programs, processes. *Theory of Computing*, 51:313–329, 2012.

[CG99] Alberto Ciaffaglione and Pietro Di Gianantonio. A co-inductive approach to real numbers. In *Proc. of the workshop "Types 1999"*, volume 1956 of *LNCS*, pages 114–130. Springer Verlag, Berlin, Heidelberg, New York, 1999.

[CG00] Alberto Ciaffaglione and Pietro Di Gianantonio. A tour with constructive real numbers. In Paul Callaghan, Zhaohui Luo, James McKinna, and Robert Pollack, editors, *TYPES*, volume 2277 of *Lecture Notes in Computer Science*, pages 41–52. Springer, 2000.

[CG06] Alberto Ciaffaglione and Pietro Di Gianantonio. A certified, corecursive implementation of exact real numbers. *Theor. Comput. Sci.*, 351(1):39–51, 2006.

[Chu11] Chi Ming Chuang. *Extraction of Programs for Exact Real Number Computation Using Agda*. PhD thesis, Swansea University, Wales, UK, 2011.

[Coq13] Coq Development Team. *The Coq Proof Assistant Reference Manual – Version 8.4pl2*. Inria, 2013.

[Fil99] Andrzej Filinski. A semantic account of type-directed partial evaluation. In *Principles and Practice of Declarative Programming 1999*, volume 1702 of *LNCS*, pages 378–395. Springer Verlag, Berlin, Heidelberg, New York, 1999.

[GHP06] N. Ghani, P. Hancock, and D. Pattinson. Continuous functions on final coalgebras. In J. Power, editor, *Proc. CMCS 2006*, Electr. Notes in Theoret. Comp. Sci., 2006.

[Goa80] Chris Goad. Proofs as description of computation. In Wolfgang Bibel and Robert A. Kowalski, editors, *CADE*, volume 87 of *Lecture Notes in Computer Science*, pages 39–52. Springer, 1980.

[Göd58] Kurt Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunkts. *Dialectica*, 12:280–287, 1958.

[GW03] Herman Geuvers and Freek Wiedijk, editors. *Types for Proofs and Programs, Second International Workshop, TYPES 2002, Berg en Dal, The Netherlands, April 24-28, 2002, Selected Papers*, volume 2646 of *Lecture Notes in Computer Science*. Springer, 2003.

[Hag87a] Tatsuya Hagino. *A Categorical Programming Language*. PhD thesis, University of Edinburgh, 1987. `http://http://www.tom.sfc.keio.ac.jp/~hagino/thesis.pdf`.

[Hag87b] Tatsuya Hagino. A typed lambda calculus with categorical type constructions. In D.H. Pitt, A. Poigné, and D.E. Rydeheard, editors, *Category Theory and Computer Science*, volume 283 of *LNCS*, pages 140–157. Springer Verlag, Berlin, Heidelberg, New York, 1987.

[HKS10] Simon Huber, Basil A. Karádais, and Helmut Schwichtenberg. Towards a formal theory of computability. In R. Schindler, editor, *Ways of Proof Theory: Festschrift for W. Pohlers*, pages 251–276. Ontos Verlag, 2010.

[HN87] Susumu Hayashi and Hiroshi Nakano. *PX: A Computational Logic*. The MIT Press, 1987.

[Hou06] Tie Hou. Coinductive proofs for basic real computation. In Arnold Beckmann, Ulrich Berger, Benedikt Löwe, and John V. Tucker, editors, *CiE*, volume 3988 of *Lecture Notes in Computer Science*, pages 221–230. Springer, 2006.

[HPG09] P. Hancock, D. Pattinson, and N. Ghani. Representations of stream processors using nested fixed points. *Logical Methods in Computer Science*, 5(3), 2009.

[Kle45] S. C. Kleene. On the Interpretation of Intuitionistic Number Theory. *The Journal of Symbolic Logic*, 10(4):109–124, 1945.

[Kre59] Georg Kreisel. Interpretation of analysis by means of constructive functionals of finite types. In Arend Heyting, editor, *Constructivity in Mathematics*, pages 101–128. North-Holland, Amsterdam, 1959.

[Let02] Pierre Letouzey. A new extraction for coq. In Geuvers and Wiedijk [GW03], pages 200–219.

[Let08] Pierre Letouzey. Coq Extraction, an Overview. In A. Beckmann, C. Dimitracopoulos, and B. Löwe, editors, *Logic and Theory of Algorithms, Fourth Conference on Computability in Europe, CiE 2008*, volume 5028 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, Heidelberg, New York, 2008.

[MFS13] Kenji Miyamoto, Fredrik Nordvall Forsberg, and Helmut Schwichtenberg. Program extraction from nested definitions. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *ITP*, volume 7998 of *Lecture Notes in Computer Science*, pages 370–385. Springer, 2013.

[Min] The Minlog System. `http://www.minlog-system.de/`.

[ML71] Per Martin-Löf. Hauptsatz for the intuitionistic theory of iterated inductive definitions. In J.E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, pages 179–216. North-Holland, Amsterdam, 1971.

[MS13] Kenji Miyamoto and Helmut Schwichtenberg. Program extraction in exact real arithmetic. *Mathematical Structure of Computer Science*, 2013. in press.

[Nup] Nuprl. `http://www.nuprl.org/`.

[Pau00] Lawrence C. Paulson. A fixedpoint approach to (co)inductive and (co)datatype definitions. In Gordon D. Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, Language, and Interaction*, pages 187–212. The MIT Press, 2000.

[Pet13] Iosif Petrakis. Advances in the theory of computable functionals TCF^+ due to its implementation. Manuscript, 2013.

[Plu98] Dave Plume. A calculator for exact real number computation. 4th Year Project Report, Departments of Computer Science and Artificial Intelligence, University of Edinburgh, 1998. `http://www.cs.bham.ac.uk/~mhe/plume.ps.gz`.

[PM89] Christine Paulin-Mohring. Extracting f(omega)'s programs from proofs in the calculus of constructions. In *POPL*, pages 89–104. ACM Press, 1989.

[Rat11] Diana Ratiu. *Refinement of Classical Proofs for Program Extraction*. PhD thesis, Mathematisches Institut der Universität München, 2011.

[RS10] Diana Ratiu and Helmut Schwichtenberg. Decorating proofs. In S. Feferman and W. Sieg, editors, *Proofs, Categories and Computations. Essays in honor of Grigori Mints*, pages 171–188. College Publications, 2010.

[Sch92] Helmut Schwichtenberg. Proofs as programs. In P. Aczel, H. Simmons, and S. Wainer, editors, *Proof Theory*, pages 81–113. Cambridge University Press, 1992.

[Sch06a] Helmut Schwichtenberg. Constructive analysis with witnesses. In H. Schwichtenberg and K. Spies, editors, *Proc. NATO Advanced Study Institute, Marktoberdorf, 2003*, volume 200 of *Series III: Computer and Systems Sciences*, pages 323–353. IOS Press, 2006.

[Sch06b] Helmut Schwichtenberg. Inverting monotone continuous functions in constructive analysis. In A. Beckmann, U. Berger, B. Löwe, and J.V. Tucker, editors, *Logical Approaches to Computational Barriers. (Proc. CiE 2006, Swansea)*, volume 3988 of *LNCS*, pages 490–504. Springer Verlag, Berlin, Heidelberg, New York, 2006.

[Sch08] Helmut Schwichtenberg. Dialectica interpretation of well-founded induction. *Math. Logic. Quarterly*, 54(3):229–239, 2008.

[Sch12] Helmut Schwichtenberg. Constructive analysis with witnesses. Manuscript, April 2012. `http://www.math.lmu.de/~schwicht/seminars/semws11/constr11.pdf`.

[Sco82] Dana Scott. Domains for denotational semantics. In Mogens Nielsen and Erik-Meineche Schmidt, editors, *Automata, Languages and Programming*, volume 140 of *LNCS*, pages 577–610. Springer, 1982.

[SW12] Helmut Schwichtenberg and Stanley S. Wainer. *Proofs and Computations*. Perspectives in Logic. Association for Symbolic Logic and Cambridge University Press, 2012.

[Tat91] Makoto Tatsuta. Program synthesis using realizability. *Theor. Comput. Sci.*, 90(2):309–353, 1991.

[Tat98] Makoto Tatsuta. Realizability of monotone coinductive definitions and its application to program synthesis. In Johan Jeuring, editor, *MPC*, volume 1422 of *Lecture Notes in Computer Science*, pages 338–364. Springer, 1998.

[Tri12] Trifon Trifonov. *Analysis of methods for extraction of programs from non-constructive proofs*. PhD thesis, Mathematisches Institut der Universität München, 2012.

[TS00] Anne S. Troelstra and Helmut Schwichtenberg. *Basic Proof Theory*. Cambridge University Press, second edition, 2000.

[vD94] Dirk van Dalen. *Logic and Structure*. Springer Verlag, Berlin, Heidelberg, New York, Berlin, 3. edition, 1994.

[Wie77] Edwin Wiedmer. *Exaktes Rechnen mit reellen Zahlen und anderen unendlichen Objekten*. PhD thesis, ETH Zürich, 1977.

[Wie80] Edwin Wiedmer. Computing with infinite objects. *Theoretical Computer Science*, 10:133–155, 1980.

[Win93] Glynn Winskel. *The Formal Semantics of Programming Languages*. Foundations of Computing Series. The MIT Press, Cambridge, Massachusetts, 1993.